وزارة التعليم العالي و البحث العلمي

BADJI MOKHTAR-ANNABA UNIVERSITY
UNIVERSITE BADJI MOKHTAR-ANNABA

جامعة باجي مختار– عنابـــــة

Faculté des Sciences de l'Ingéniorat Année 2014-2015

Département d'Informatique

THESE

Présentée en vue de l'obtention
du diplôme de Doctorat en Sciences

DEMARCHE DE DEVELOPPEMENT DES SYSTEMES ONTOGENETIQUES

Option
Génie Logiciel

Présentée par

Mme KHERISSI Farida

Directeur de Thèse
Mr Djamel MESLATI Prof. Université Badji Mokhtar-Annaba
Co-directrice de Thèse
Dr Dalila TAMZALIT MC Université de Nantes-France

DEVANT LE JURY

Président
Mr Salim GHANEMI Prof. Université Badji Mokhtar-Annaba
Examinateurs
Mr Zarour Nacer Eddine Pr Université de Constantine 2
Mr Zine Eddine Bouras MCA EPST Annaba

REMERCIEMENTS

Je tiens à exprimer toute ma gratitude à Mr MESLATI Djamel, à la fois Directeur du laboratoire LISCO et mon directeur de thèse, pour la confiance qu'il m'a témoignée et pour sa haute qualité d'encadrement tout au long de ces années. Ses conseils, sa rigueur et ses qualités humaines ont largement contribué à l'aboutissement de cette thèse. Je tiens à lui exprimer toute ma reconnaissance et tout mon respect.

Je remercie également TAMZALIT Dalila, ma Co-directrice de thèse, qui m'a été d'une aide considérable avec ses conseils pendant mes séjours au laboratoire LINA de l'université de Nantes.

J'adresse mes vifs remerciements au Prof. GHANEMI Salim qui m'a fait l'honneur de présider le jury de soutenance de cette thèse.

Je tiens à exprimer mes sincères remerciements au Prof. ZAROUR Nacer Eddine et au Dr BOURAS Zineddine pour avoir accepté d'examiner mon travail.

Je tiens aussi à remercier tous mes collègues du laboratoire LISCO, en particulier Mme BOUNOUR-ZEGHIDA Nora pour ses conseils et ses remarques qui m'ont été utiles, Mme SOUICI-MESLATI Labiba pour ses encouragements. Je tiens aussi à remercier mes collègues de la FSEG, en particulier Mme BOUCHEMAL Najette pour ses encouragements et son esprit de fraternité, Mme Oualhi Ouarda, sans oublier Mme Khayari Zahia et les autres collègues.

J'adresse aussi mes remerciements aux enseignants du département d'informatique dont la majorité sont actuellement à l'étranger et qui ont contribué à ma formation depuis mon premier cycle universitaire.

Enfin, je rends hommage à toute ma petite et grande famille, en particulier ma sœur Latifa.

DEDICACES

A la mémoire de mon père

TABLE DES MATIERES

TABLE DES ILLUSTRATIONS

INTRODUCTION

Contexte de recherche

Faire face à l'évolution des systèmes logiciels demeure actuellement un problème majeur du génie logiciel. En effet, dès que le logiciel est livré, il débute une longue phase de maintenance qui est censée faire face aux changements des besoins des utilisateurs, améliorer les performances, l'adapter aux changements de la plateforme matérielle ou simplement corriger les erreurs non détectées durant la phase de test.

Faire évoluer un logiciel est souvent problématique pour diverses raisons. Au-delà du fait qu'il faut disposer d'une documentation adéquate du logiciel pour le comprendre, il faut aussi opérer une phase de reverse engineering qui vise à retrouver des décisions de conception non apparentes dans la documentation ou le code du logiciel. Sans une bonne compréhension, les changements opérés peuvent engendrer d'autres changements envahissants, altérer l'architecture du logiciel (on parle alors de dérive architecturale) ou ne plus garantir diverses propriétés importantes. Ce qui a pour conséquence d'altérer la qualité du logiciel et conduire à son retrait prématuré.

Bien que l'évolution constitue la phase la plus couteuse du cycle de vie du logiciel, elle est restée longuement menée par des approches ad hoc où règnent l'improvisation et le pragmatisme. L'approche des systèmes ontogénétiques aborde l'évolution d'une manière qui s'inspire de l'ontogénèse des systèmes biologiques. Son point de départ consiste à concevoir dès le début un système logiciel pour évoluer. Autrement dit, le logiciel n'est plus une infrastructure instantanée relative à une réalité donnée, à un moment donné, mais une infrastructure qui évolue continuellement et de manière autonome et cela dès sa création initiale.

Cette vision radicale de l'évolution sépare les changements possibles en deux groupes : les changements anticipés et les changements non anticipés.

Les changements anticipés sont les changements qu'on peut prévoir alors que le logiciel est en cours de développement. D'habitude, de tels changements sont pris en compte lors du développement et incorporés dans le code du logiciel moyennant une chaîne d'instructions alternatives éventuellement imbriquées (if … then…else… if … then…else…). Ce qui altère inévitablement les performances du logiciel.

L'approche ontogénétique reporte la prise en compte des changements anticipés jusqu'à ce qu'ils deviennent nécessaires (i.e. certaines conditions se réalisent), suite à cela le code est altérer dynamiquement pour incorporer ces changements. Cette approche a l'avantage de maintenir le code performant, simple et reflétant les besoins réels actuels. Concrètement le code est débarrassé des structures alternatives au profit d'une évolution dynamique des comportements des objets en fonctions des besoins réels actuels. Bien entendu cela suppose que le code du logiciel ait une structuration qui permet d'incorporer les changements de manière adéquate et que la plateforme d'accueil soit dotée de composants qui peuvent altérer le code du logiciel dynamiquement en fonction de la description du changement anticipé.

Les changements non anticipés sont des changements qui apparaissent une fois le logiciel livré. L'approche ontogénétique traite les deux types de changement de la même façon au niveau opérationnel. Cependant au niveau abstrait, le traitement diffère. La démarche utilisée doit tenir compte d'un existant qu'il faut comprendre, dont-il faut respecter certaines propriétés, etc.

La prise en compte de l'ontogenèse constitue un nouveau défi et une vision radicale de l'évolution qui a un effet aussi bien sur notre façon de percevoir les systèmes logiciels que notre manière de les concevoir. Nous nous intéressons dans cette thèse à la proposition d'une démarche pour le développement des systèmes ontogénétiques. Etant donné que la création d'une nouvelle méthode dédiée à ces systèmes est une tâche fastidieuse et demande des efforts qui outrepassent le contexte académique dans lequel nous travaillons, nous avons opté pour l'extension d'une méthode existante. Afin de faire face à la complexité croissante du développement logiciel, des méthodes de gestion des projets informatiques ont été introduites. Celles dites traditionnelles sont axées-plan et généralement lourdes, ces méthodes ont perdu de plus en plus de partisans en faveur des nouvelles méthodes dites « agiles ».

Nous avons étudié plusieurs méthodes agiles afin de choisir la plus apte à une extension supportant les systèmes ontogénétiques et nous avons opté pour l'extension du Rational Unified Process. Le RUP est l'une des méthodes agiles basées sur des pratiques du génie logiciel avérées et sur le formalisme UML (Unified Modeling Language), qui présente un méta-modèle d'architecture extensible et personnalisable [Kru 03].

Notre travail dans cette thèse, concerne l'étude des opportunités offertes pour l'ontogénèse par le processus RUP et le formalisme UML en termes d'éléments de modélisation ou artéfacts qui sont créés et maintenus dans la démarche et font la base du développement de systèmes logiciels. L'approche que nous proposons, pour la prise en compte de l'ontogenèse, étend principalement le processus d'ingénierie des besoins de changements/évolutions anticipés, et propose également une discipline pour la prise en charge des évolutions non anticipées et la mise à jour dynamique d'un système.

Problématique

L'approche ontogénétique affecte toutes les phases du cycle de vie d'un logiciel. Le support d'une telle approche demeure un défi considérable. L'approche nommée Mage donne une implémentation possible de l'approche ontogénétique [Mes 06]. Cependant, comme souligné dans [Khe 09a], le défi de l'approche se situe plutôt au niveau de la démarche de développement. En effet, l'approche soulève de nombreuses question : Comment peut-on déduire les changements anticipés ou non anticipés ? Comment analyse-t-on l'état actuel d'un système et après plusieurs changements ? Comment garantir formellement que l'introduction d'un changement préserve les propriétés invariantes d'un système ? Comment préserver l'architecture d'un système ? Comment garantir le maintien des qualités de services ? Existent-ils des méthodes de développement qui pourront résoudre ou répondre à ces questions ?

Malgré l'importance de l'approche ontogénétique, il n'existe aucune méthodologie de développement pouvant la supporter.

En effet, tout changement des besoins est géré avant la livraison du produit final. Appréhender préalablement le changement, c'est-à-dire permettre de concevoir le système pour changer, est une vision non prévue dans toutes les méthodes de développements de systèmes logiciels, qu'elles soient traditionnelles ou agiles. En particulier, les approches agiles tiennent compte du changement durant le développement mais elles ne sont pas armées pour créer de systèmes ontogénétiques. En général, dans les méthodes existantes, le processus de changement est assuré par l'intervention de l'être humain sur un système logiciel pour le maintenir conforme au domaine qu'il représente et répondant aux besoins évolutifs des utilisateurs. En conséquence, la problématique, dans ce contexte, demeure la recherche d'un cadre méthodologique pour supporter les systèmes ontogénétiques.

Motivations

Le nombre d'opérateurs impliqués dans l'utilisation des logiciels, les enjeux qui en découlent ainsi que les ressources énormes engagées dans la production et la maintenance de logiciels sont des facteurs qui justifient et motivent les travaux de recherche sur l'évolution, les problèmes qu'elle cause et les solutions possibles.

L'approche ontogénétique est un paradigme, qui offre une nouvelle vision de l'évolution. Les systèmes logiciels ontogénétiques présentent la caractéristique d'évoluer dynamiquement et de manière autonome pour répondre aux besoins des utilisateurs et leurs changements qu'ils soient anticipés ou non. L'évolution de tels systèmes est perçue comme un processus qui les façonne dès leur création. Pour tirer profit de tels systèmes, il

est nécessaire d'investir dans une démarche de développement appropriées qui redonnent à l'évolution le statut de concept central.

Objectifs

Le travail de cette thèse se situe au carrefour de plusieurs domaines : l'ingénierie des méthodes de développement de systèmes logiciels, ingénierie des besoins, et la modélisation de l'évolution des besoins. Nous nous sommes fixés comme objectifs l'extension du processus RUP pour supporter les systèmes ontogénétiques.

Contenu du mémoire

Cette thèse se compose de cinq chapitres, de cette introduction et de la conclusion générale. Le partage que nous avons adopté reflète, à la fois, notre démarche de travail durant cette thèse et un ordre intelligible.

Chapitre 1. Ce chapitre traite de l'ontogénèse comme une nouvelle vision de l'évolution des systèmes logiciels. On y présente aussi l'approche ontogénétique Mage.

Chapitre 2. Dans ce chapitre nous discutons l'existant concernant les processus de développement de systèmes logiciel. Nous y décrivons particulièrement le processus Rational Unified Processus que nous avons choisi d'étendre.

Chapitre 3. Dans ce chapitre nous présentons une étape clé dans le cycle de vie des systèmes logiciels et ontogénétiques, il s'agit de l'ingénierie des besoins. Nous donnons des concepts et approches existants dans la littérature et nous terminons par les paradigmes et approches utilisés dans le processus d'ingénierie proposé dans notre démarche.

Chapitre 4. Décrit l'approche proposée pour le support du développement des systèmes ontogénétiques.

Chapitre 5. Présente des exemples d'application pour montrer le scénario de développement dans le cadre de la démarche proposée et donne un aperçu sur l'aspect implémentation.

CHAPITRE 1
LES SYSTEMES ONTOGENETIQUES ET MAGE

1.1. Introduction

En analysant les organismes vivants, on peut constater deux types d'évolutions. Une évolution qui façonne l'organisme de sa naissance à sa mort et une évolution qui tend à différencier les descendants d'un organisme par rapport à ce dernier. Les systèmes biologiques renferment des mécanismes et des processus longuement éprouvés qui leur ont permis de survivre face à un environnement changeant et contraignant. Ce fait a conduit l'être humain à les considérer comme source d'inspiration dans la conception des systèmes artificiels. Ce chapitre s'intéresse aux processus naturels dédiés aux changements et à l'évolution, et présente l'approche Mage qui modélise l'ontogenèse d'un système logiciel sous la forme d'un génome embarqué, dont le rôle consiste à façonner continuellement le système en fonction des changements anticipés et non anticipés.

1.2 Facettes de l'évolution

Il est actuellement communément admis que les organismes biologiques sont façonnés par trois processus ou trois niveaux d'organisation : ontogenèse, épigenèse et phylogenèse. Ces trois processus, connu aussi sous l'appellation du modèle POEtic (appellation issue de Phylogenesis, Ontogenesis et Epigenesis) ont été identifiés dans le cadre du projet " Reconfigurable POEtic Tissue", lequel projet fut, conduit sous l'égide du programme européen des technologies de la société d'information (European Program of Information Society Technologies) [Mes 06]. Nous présentons dans ce qui suit les trois processus puis nous discutons des métaphores qui en sont issues.

1.2.1 Ontogenèse

Les organismes vivants ne naissent pas complètement formés tels que nous les voyons. L'organisme commence sa vie en tant que cellule unique, dotée d'un programme de développement codé dans son génome. Ce dernier est continuellement exécuté par la

cellule, ce qui conduit à sa division répétée engendrant une multitude de cellules identiques ayant le même programme [Mes 06]. Suite à cela, une forme de communication s'opère entre les cellules permettant à chacune d'exécuter la partie du programme du génome qui correspond à sa position dans l'ensemble. Cette différenciation cellulaire conduit, en fin de compte, à la formation d'organes et donne à l'individu la morphologie et le comportement spécifiques à son espèce. Le processus de développement qui façonne un organisme tout au long de sa vie est appelé ***ontogenèse***. L'ontogenèse est un processus déterministe dont l'exécution est influencée par l'environnement où elle se déroule.

1.2.2 Phylogenèse

Dans une espèce donnée, la reproduction consiste à transmettre le génome d'un ou deux ascendants vers le descendant. Le génome de la première cellule du descendant est obtenu de celui de ces ascendants par mutation et entrecroisement. Etant différent de celui de ces ascendants, le génome contrôle l'ontogenèse et produit un organisme différent des ascendants. Grâce au changement du génome, le descendant acquière de nouvelles propriétés dont dépend sa survie. La mutation et l'entrecroisement produisent progressivement un changement et une évolution de l'espèce d'une génération à l'autre. Cette évolution est appelée *phylogenèse*. La phylogenèse est un processus non déterministe qui n'a aucun effet sur l'organisme lui-même mais en a sur l'espèce. La phylogenèse introduit la diversité dans les organismes vivants et elle est importante pour leur survie, leur adaptation continue à l'environnement ainsi que l'apparition de nouvelles espèces. Le processus phylogénétique est basé sur la sélection naturelle qui permet la survie des individus adaptés à leur environnement. Par conséquent, à travers la phylogenèse, l'environnement peut avoir un impact majeur sur l'évolution des espèces [Mes 06].

1.2.3 Epigenèse

Vu que le génome est limité dans la quantité d'informations qu'il peut stocker, et vu que l'altération du génome par l'environnement à travers l'ontogenèse et la phylogenèse est lente et limitée, les organismes complexes sont façonnés par un troisième processus appelé *épigenèse*. Ce dernier utilise des structures spécifiques (dans l'individu) pour stocker et gérer un nombre important d'interactions avec l'environnement. Le processus épigénétique est supporté par les trois systèmes : le *système nerveux*, le *système endocrinien* et le *système immunitaire*. Les structures utilisées par ces systèmes sont facilement altérables et permettent aux organismes vivants complexes d'apprendre et d'effectuer un traitement symbolique de l'information [Mes 06].

1.2.4 Métaphores Biologiques et processus d'Evolution

On entend par métaphore biologique une analogie qu'on cherche à déterminer entre le monde biologique et le monde artificiel, de façon à pouvoir proposer des approches qui imitent certains aspects du premier, tout en en ignorant d'autres. Fondamentalement, les métaphores ne cherchent pas à reproduire ce qui est biologique, mais plutôt à l'interpréter en fonction de ce qu'il est possible et raisonnable de réaliser. De ce fait, on peut déduire que les métaphores biologiques sont évolutives et dépendent de notre compréhension de la réalité et notre aptitude à en extraire des éléments pratiques et bénéfiques.

Les tendances actuelles des systèmes bio-inspirés proposent de nouvelles métaphores biologiques et leur combinaison (i.e. hybridation) pour construire des systèmes exhibant les propriétés désirables comme les comportements émergeants, l'adaptabilité à l'environnement, la cicatrisation, etc. Nous décrivons ci-après ces processus et les métaphores qui en ont été déduites.

Ontogenèse et métaphores. Le processus de développement qui façonne un organisme tout au long de sa vie est appelé ontogenèse. L'ontogenèse est un processus déterministe dont l'exécution est influence par l'environnement où elle se déroule. Concernant l'ontogenèse, la première tentative d'inspiration fut celle de Von Neumann avec sa machine qui s'auto duplique (self replicating machine). Cette dernière est un automate capable d'effectuer un calcul universel (i.e. équivalent à la machine de Turing) et une construction universelle. Les métaphores courantes tentent de mimer d'autres mécanismes de l'ontogenèse comme la division ou la différentiation cellulaire [Mes 06].

Phylogenèse et métaphores. La métaphore phylogénétique courante est celle des algorithmes génétiques et leurs diverses variantes. D'une manière générale, les algorithmes génétiques imitent la phylogenèse en résolvant des problèmes d'optimisation. Les solutions candidates d'un problème donné sont considérées comme des individus et codées sous forme d'un génome ayant la forme d'une chaîne de symboles abstraits. Les individus dans l'espace des solutions candidates sont ensuite évolués par des opérations de mutation et d'entrecroisement puis sélectionnés d'une génération à l'autre moyennant une fonction d'adaptation (i.e. un critère de qualité donné). Cette procédure itérative continue jusqu'à ce qu'un nombre d'itération soit achevé ou qu'aucune amélioration sensible n'est enregistrée. Malgré la simplicité et la nature aveugle de l'approche, elle a été utilisée avec succès à travers un large éventail d'applications [Mes 06].

Epigenèse et métaphores. Des métaphores de l'épigenèse (système nerveux, système immunitaire et système endocrinien) sont actuellement utilisées. Le système nerveux fût le premier à être étudié et à recevoir un maximum d'attention, ce qui a donné naissance au domaine des réseaux de neurones artificiels. La métaphore des réseaux de neurones

artificiels est une tentative à mimer les caractéristiques fondamentales des neurones biologiques. Les réseaux de neurones artificiels peuvent être vus comme des graphes dirigés avec des connections pondérées entre les neurones (i.e. les synapses). La possibilité d'apprendre par des processus d'apprentissage dans le contexte des réseaux de neurones artificiels est vue comme un problème d'ajustement de l'architecture du réseau et des poids des connexions, de telle sorte à ce que le réseau puisse effectuer une tâche spécifiques de façon performante. Les réseaux de neurones artificiels sont convenables pour les problèmes où la compréhension est limitée ou incomplète et où existe une abondance de données. C'est le cas typique du problème de reconnaissance de formes. Le succès des réseaux de neurones artificiels est comparable à celui des algorithmes génétiques ; toutefois leur domaine d'application est relativement limité [Mes 06].

En appliquant les processus précédents, l'aspect structurel de l'organisme peut être vu comme composé d'un génome et d'un phénotype. Ce dernier comprend les propriétés dérivées du génome en utilisant n'importe lequel des processus Poetic. Le phénotype a donc une partie innée dérivée par l'ontogenèse et une partie acquise dérivée par épigenèse. Les deux étant façonnés à long terme par la phylogenèse.

Actuellement, le processus de changement est vu principalement comme l'intervention de l'être humain sur un système pour le maintenir conforme au domaine qu'il représente et répondant aux besoins évolutifs des utilisateurs. En effet, dans l'activité de maintenance, on considère un modèle comme une entité passive ou factuelle. Lorsqu'on modélise le processus ontogénétique, on appréhende préalablement le changement : on conçoit le système pour changer. L'ontogenèse est vue comme une nouvelle dimension des systèmes logiciels qui est orthogonale à leurs dimensions structurelle et comportementale. En tant que nouvelle dimension, l'ontogenèse présente la caractéristique d'englober tout type de changement qu'un système subit, pendant son développement et sa maintenance.

Le changement de façon autonome est une caractéristique souhaitable dans les systèmes adaptatifs qui doivent s'accommoder d'environnements d'exécution différents. L'autonomie implique la mise à jour dynamique d'un système. C'est une caractéristique souhaitable pour les systèmes embarqués et nécessaire dans les systèmes critiques qui ne peuvent tolérer un arrêt d'exécution. On cite : les systèmes de contrôle du trafic aérien et ferroviaire, la gestion des commutations téléphoniques, la gestion des transactions financières et la gestion des centrales téléphoniques, la gestion des transactions financières et la gestion des centrales électriques [Mes 06].

1.3 L'ontogénèse biologique

1.3.1 Définitions

Un être vivant est un organisme qui présente une forme bien définie, qui est animé, qui respire, s'alimente, se développe et se reproduit. Nous entendons par système biologique, un organisme isolé aussi bien qu'un ensemble composé de plusieurs organismes vivants, semblables ou différents. L'étude des systèmes biologiques est une tâche difficile, car un être vivant pris isolement, présente déjà une grande complexité. En effet, dans sa forme la plus élaborée, l'organisme vivant est composé d'organes liés et ordonnés de façon à assurer sa survie et sa reproduction. Plusieurs être vivants peuvent s'assembler dans un système où règnent un ordre et des interactions complexes. Dans la nature, tout système est sujet à l'évolution.

Il existe plusieurs définitions de l'évolution biologique. Bien qu'elle désigne un nombre important de concepts, les définitions existantes vont dans le même sens : c'est-à-dire le changement à long terme des espèces.

Dans l'encyclopédie Universalis, l'évolution est définie par [Uni 14a] :

Le processus par lequel, au cours des âges, se succèdent et s'engendrent, tout en variant, les espèces végétales et animales.

Dans [Rid 96], on trouve la définition suivante :

Evolution signifie changement dans la structure et le comportement des organismes, au fil des générations. Les aspects divers des organismes actuels, à tous les niveaux depuis la séquence de leur ADN jusqu'à leurs structures macroscopiques ou jusqu'à leur comportements sociaux, sont issus de la modification de ceux de leur ancêtres.

1.3.2 Développement des organismes

Dès sa naissance, et même avant, un organisme se caractérise par un ensemble de propriétés que l'être humain peut percevoir par ses organes sensoriels ou par ses capacités d'analyse et de déduction. Cet ensemble de propriétés, appelé ***phénotype***, renferme aussi bien des propriétés physiques que comportementales. Par exemple, une plante peut être caractérisée par sa taille, sa morphologie, sa saison de floraison, son mécanisme de reproduction. De même, un animal peut inclure dans son phénotype des propriétés telles que la couleur, le poids, la possession des ailes, la stratégie de défense contre les prédateurs, etc.

Durant sa vie, un organisme change continuellement son phénotype et passe d'une phase à une autre. Le développement désigne ce processus de changement continuel et couvre deux fonctions consistant à engendrer la diversité et à assurer la continuité de la vie d'une génération à la suivante.

1.4 Vers des Systèmes Ontogénétiques

Plusieurs applications importantes doivent s'exécuter continuellement et sans interruption. C'est le cas des systèmes critiques, tels que la commutation téléphonique, les transactions financières, la réservation des places d'avion et le contrôle du trafic aérien ou ferroviaire. Un autre facteur vient augmenter le rang des applications critiques, il s'agit du développement rapide de l'utilisation du Web et d'Internet pour différents services. Ainsi, pour diverses applications, telles que le commerce électronique, l'arrêt total est préjudiciable : perte de certains états accumulés (états non persistants) et annulation des traitements en cours, pouvant engendrer des incohérences mal perçus par les utilisateurs de ces applications. Les systèmes mobiles représentent une catégorie de systèmes qui peuvent tirer un avantage certain de la mise à jour dynamique car elle leur permet une adaptation rapide aux changements fréquents de leur contexte [Mes 06].

Par ailleurs, il y a nécessité de changer ces applications pour permettre l'élimination de certains bugs, l'ajout/amélioration des services qu'elles offrent, le changement de plateformes, etc. De ce fait, une mise à jour dynamique est nécessaire. Par dynamique, on sous entend son déroulement pendant l'exécution de l'application. La mise à jour dynamique est un besoin ancien, cependant, les réponses qui ont été apportées jusqu'a ce jour sont, dans leur grande majorité, spécifiques aux applications [Hic 01].

D'une manière générale, la mise à jour dynamique se définit par la mise à jour d'un système logiciel pendant son exécution pour modifier son code ou ses données [Bie 03].

Bien que les systèmes conçus par l'homme ressemblent de plus en plus aux systèmes biologiques, la ressemblance n'est qu'apparente. Sans entrer dans un long débat sur les caractéristiques de la vie et du vivant, on peut facilement réunir sous l'intitulé « inaptitude à l'évolution » les caractéristiques de nos systèmes artificiels.

En effet les organismes biologiques sont autonomes, adaptatifs vis-à-vis de leur environnement, se reproduisent, se développent et guérissent suite aux maladies, se régénèrent et se cicatrisent suite aux blessures, résistent aux attaques et développent des mécanismes de défense, etc.

A l'opposé, nos systèmes artificiels logiciels ou autres, ne présentent que superficiellement quelques unes de ces propriétés. Nos systèmes les plus complexes peuvent s'arrêter de façon nette si le bit du code d'une instruction est altéré. Nos circuits électroniques les plus sophistiqués peuvent faillir à leurs fonctions suite à une augmentation ou diminution de température. Les systèmes hautement critiques peuvent subir des attaques virales sans qu'il y apparaisse une réaction convenable qui préserve leurs fonctions essentielles, etc.

Un système logiciel ontogénétique est un système doté d'un mécanisme qui fait évoluer ses propriétés statiques et comportementales de manière autonome, dynamique et continue en réponse à des stimuli internes et cela conformément à une description de l'évolution préalablement établie et embarquée dans le système lui-même.

Cette définition entraine des caractéristiques qui différencient un système classique d'un système ontogénétique. Nous discutons ces points dans ce qui suit :

- **Evolution des propriétés statiques et comportementales**

Modéliser un système revient à modéliser sa structure statique et son comportement. Dans les approches classiques les comportements font évoluer le contenu des variables qui représentent l'aspect statique du système mais ne font pas évoluer le type de ces variables ou leur nombre. De même les comportements ne sont pas sujets au changement. Dans les systèmes ontogénétiques, il est question de faire évoluer la structure et le comportement d'un système à l'image de ce que ferait un agent de maintenance.

- **De manière autonome**

Le système évolue de manière autonome sans intervention externe. Ceci revient à redonner au système une place plus active en lui affectant une charge plus importante dans les tâches qui le font évoluer.

- **De manière dynamique**

Le système évolue tout en restant opérationnel.

- **De manière continue**

L'évolution a lieu à tout moment. C'est un processus continu qui façon le système depuis sa création.

- **En réponse à des stimuli internes**

Faire évoluer un système vient suite à un besoin quelconque qui se manifeste par des stimuli qui émanent du système lui-même.

- **Description de l'évolution préalablement établie**

Toute opération d'évolution est décrite préalablement sous forme d'un patch puis injectée dans le mécanisme qui fait évoluer le système.

- **Embarquée au sein du système lui-même**

La description de l'évolution est stockée sous une forme ou une autre dans le système.

Ces caractéristiques sous entendent que :

Un système ontogénétique se compose de deux parties, une partie opérationnelle qui exécute des fonctionnalités et une partie qui prend en charge l'évolution continue de la première.

L'évolution prend un statut central dans le sens où le système est conçu pour changer.

Le système est doté de moyens qui lui permettent d'appréhender les stimuli de l'évolution.

Le processus de l'évolution est modélisé d'une façon ou d'une autre moyennant des concepts spécifiques.

Toutes les formes d'évolution sont traitées de la même façon. Ce point est important car les systèmes ontogénétiques présentent l'avantage de considérer les deux formes de changements (anticipés ou non) de manière identique.

1.5 Le Modèle Ontogénétique Mage

1.5.1 Principes

Mage est une approche basée sur quatre principes :

- Tout changement qu'un système subit fait partie de son processus ontogénétique.
- L'ontogenèse est un processus continu. Les changements sont déclenchés dynamiquement par des événements internes et externes.
- L'ontogenèse est une dimension embarquée. Le système se développe de façon autonome.
- Les changements anticipés sont codés dans le génome et s'exécutent une fois les conditions de déclenchement atteintes. Les changements non anticipés sont codés puis injectés dans génome selon le besoin.

1.5.2 Motivations

L'approche Mage est motivée par un ensemble de propriétés dont on souhaite doter les systèmes logiciels. Nous les décrivons ci-après.

1.5.2.1 Modélisation du processus ontogénétique

La modélisation du processus ontogénétique implique deux aspects : l'intégration du processus dans le modèle et sa modélisation comme étant un processus continu.

Actuellement, le processus de changement est vu principalement comme l'intervention de l'être humain sur un système pour le maintenir conforme au domaine qu'il représente et répondant aux besoins évolutifs des utilisateurs. En effet, dans l'activité de maintenance, on considère un modèle comme une entité passive ou factuelle. Lorsqu'on modélise le processus ontogénétique, on appréhende préalablement le changement : on conçoit le système pour changer. Par modélisation, nous entendons : investir le futur d'un système pour déterminer ce qu'il deviendra, quand et sous quelles conditions il doit changer, puis ajouter un mécanisme au modèle du système, de telle sorte qu'il devienne capable de changer de façon autonome.

On peut considérer que les modèles orientés objet actuels ne sont pas factuels, car ils englobent le comportement des objets. Ceci est particulièrement vrai lorsqu'on les compare avec les modèles relationnel ou sémantique. Cependant, un objet qui possède un certain comportement, ou une certaine structure, correspond à un fait qui est susceptible de changer dans le temps. La modélisation de ce changement échappe aux approches actuelles : c'est la dimension manquante. Cette dimension est plus complexe que celles structurelle ou comportementale. De plus, elle leur est orthogonale et englobante.

Avec l'évolution des paradigmes de modélisation, nous sommes passés de la modélisation des structures statiques à la modélisation des comportements par l'intégration de ces derniers dans le modèle. En faisant cela, l'approche de modélisation est passée d'une vue statique à une vue dynamique du monde réel. Dans ce travail, nous suggérons d'aller plus loin dans cette direction en intégrant le processus de changement dans le modèle lui-même. Notons qu'aboutir à un tel modèle d'une réalité est une tâche difficile, qui demande une investigation plus profonde que celle qui consiste à extraire le comportement des objets. En effet, l'expérience avec le modèle orienté objet nous a montré que lorsqu'on passe d'une vue statique à une vue dynamique d'un système complexe, l'effort de modélisation devient considérable [Rum 96]. Cependant, l'intégration de l'aspect dynamique a permis d'obtenir des modèles plus proches de la réalité et cette proximité élimine de nombreux problèmes de maintenance. Nous prévoyons des retombées positives par le passage vers les modèles ontogénétiques.

1.5.2.2 Réduction des interactions avec le monde réel

Puisqu'elles impliquent souvent l'être humain, elles sont génératrices d'erreurs et, de ce fait, leur réduction est souhaitable. On distingue deux types d'interactions. Premièrement, les interactions qui visent à changer le modèle, elles sont réduites dans Mage, vu que les

changements sont anticipés et codés dans le génome : le système évolue de façon autonome. Les interactions correspondant aux changements non anticipés demeurent et nécessitent un changement du génome.

Deuxièmement, certaines interactions proviennent de l'utilisation des fonctionnalités du modèle : entrées, sorties et perception de stimuli externes. Même si ces interactions dépendent de la conception, elles peuvent être réduites si le modèle est toujours proche de la réalité. Par exemple, si un attribut n'est plus nécessaire, sa suppression évite de continuer à lire ou écrire sa valeur.

1.5.2.3 Réduction de l'effort de maintenance

Bien qu'un système ontogénétique nécessite moins d'interactions externes pour changer, ce n'est pas une raison directe de réduction de l'effort de maintenance. En effet, il faut tenir compte du fait, qu'au départ, on consent un effort important pour modéliser le processus de changement. La réduction vient plutôt du fait que l'anticipation des changements futurs permet d'éliminer certains changements inappropriés et les retours arrière correspondants. Par exemple, si une analyse conduit à la conclusion que, dans une certaine phase de la vie d'un objet, un type d'attribut doit être réel, on peut, éventuellement, éviter des changements intermédiaires vers les types inclus dans le type des réels.

1.5.2.4 Amélioration des performances

Elle découle des possibilités de spécialisation de programmes et d'adaptation dynamique [Kis 03, Shu 03]. Pour appréhender cela, il faut comparer un système Mage avec un *`système stable`*. Par stable, nous entendons un système qui s'accommode des changements anticipés sans maintenance et qui peut tourner sur plusieurs plateformes. Par exemple, on peut imaginer une classe qui regroupe toutes les propriétés et prévoir des instructions conditionnelles qui évitent une utilisation incorrecte (par exemple la chenille ne peut voler, telle plateforme n'a pas de coprocesseur dédié au traitement en virgule flottante, etc.). Tenir compte de toutes les situations d'incompatibilité rend le système stable non efficace, complexe et enchevêtré. Il est préconisé, dans Mage, la mise à jour dynamique d'un système et l'utilisation d'un mécanisme d'exception général (inspiré de celui de Java), pour tenir compte de l'existence d'objets à différentes phases de leur vie. De même, il est possible d'adapter dynamiquement un modèle Mage à sa plateforme d'exécution.

1.5.2.5 Implémentation dynamique de la variabilité

L'approche Mage considère que la variabilité ne se termine pas une fois le logiciel livré car, suite à une certaine période d'utilisation, d'autres éléments de différence peuvent apparaître. Mage préconise que la variabilité est en partie postérieure à la livraison car ce qu'impose l'environnement sur une famille de produits ne peut être totalement anticipé [Cop 98, Gur 01]. Par exemple : Quel type de périphériques le système doit-il gérer ? Quel type de fonctionnalités doit être en arrière plan pour telle ou telle catégorie de clients ? etc., sont des questions pour lesquelles la réponse peut ne pas exister avant la livraison du produit. Une variabilité prise en charge par le système lui-même, une fois livré, peut tenir compte d'un nombre important de détails et simplifie la tâche des développeurs.

1.5.2.6 Création d'une représentation convenable pour la phylogenèse

L'évolution en génie logiciel est effectuée actuellement par le changement manuel et par la réutilisation. Si on souhaite améliorer un composant en utilisant les algorithmes évolutionnistes, nous devons coder ce composant comme un individu et déduire une fonction d'adaptation qui sélectionne le meilleur après reproduction, croisement et mutation (meilleur peut signifier performant, convivial ou toute autre qualité). Il est communément admis que l'une des tâches les plus difficiles dans l'approche évolutionniste est le codage des individus [Ada 98, Heu 94]. Dans ce contexte, le génome d'un modèle Mage constitue un codage qui nous semble convenable pour l'utilisation de l'approche évolutionniste en génie logiciel.

Selon Dellaert, la majorité des approches phylogénétiques utilisent un mapping direct entre génome et phénotype. Il existe alors une correspondance bijective entre les chaînes de bits du génome et les propriétés phénotypiques du système ou individu qu'on souhaite améliorer. Cette approche présente un inconvénient majeur, celui de la non convergence des algorithmes dès que le nombre de propriétés devient consistant [Del 96]. Dellart ajoute « When interpreted not as a direct encoding but as a set of developmental rules, a genetic description can lead to much more complex morphologies than those achievable with direct mapping ».

1.5.2.7 Réutilisation de l'évolution

Il est communément admis que la meilleure documentation qui permet de comprendre un logiciel est son code source. De ce fait, la meilleure étude de l'évolution subie par un système est celle effectuée sur son code source, changement après changement. L'étude de l'historique de l'évolution d'un système est grandement simplifiée si le système garde lui-même la trace des changements qu'il subit. Cette étude peut éviter certaines erreurs

lors du développement des nouveaux systèmes, comme elle aide à anticiper les changements futurs.

1.5.3 Mage : Les Concepts

Un modèle Mage est composé de deux parties un génome et un phénotype. Le phénotype représente l'équivalent d'un système logiciel classique. Le génome est une collection de gènes qui façonnent continuellement le phénotype. La figure 1.1 résume l'approche de Mage vis à vis de l'évolution.

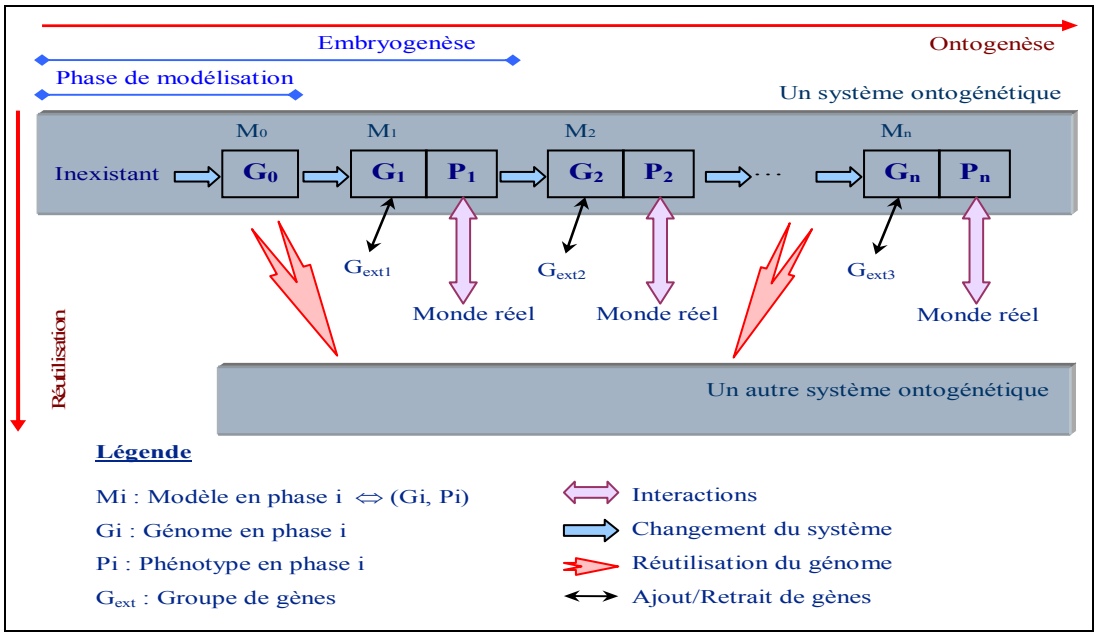


Figure 1.1. L'évolution selon Mage

Selon cette figure, on constate l'existence de trois grandes phases :

La modélisation. Un modèle commence à exister à la fin de la phase de modélisation qui consiste en une conception et une implémentation. Le but de cette phase est de produire un génome qui, une fois exécuté, donne au modèle son premier phénotype.

L'embryogenèse. La fin d'une phase importante du modèle est atteinte lorsqu'il parvient à la fin de l'embryogenèse. Cette fin est marquée par l'aptitude du modèle à interagir avec l'environnement (dans le sens de l'exécution des fonctions).

L'évolution continue. Le génome demeure actif et façonne continuellement le modèle. Les changements non anticipés sont injectés sous forme de gènes dans le génome.

Durant la phase de l'évolution continue, un modèle Mage se présente sous la forme donnée en figure 1.2. Cette dernière fait ressortir quatre notions : le phénotype, le génome, l'interaction et les stimuli. Nous décrivons ci-après le rôle de chacune.

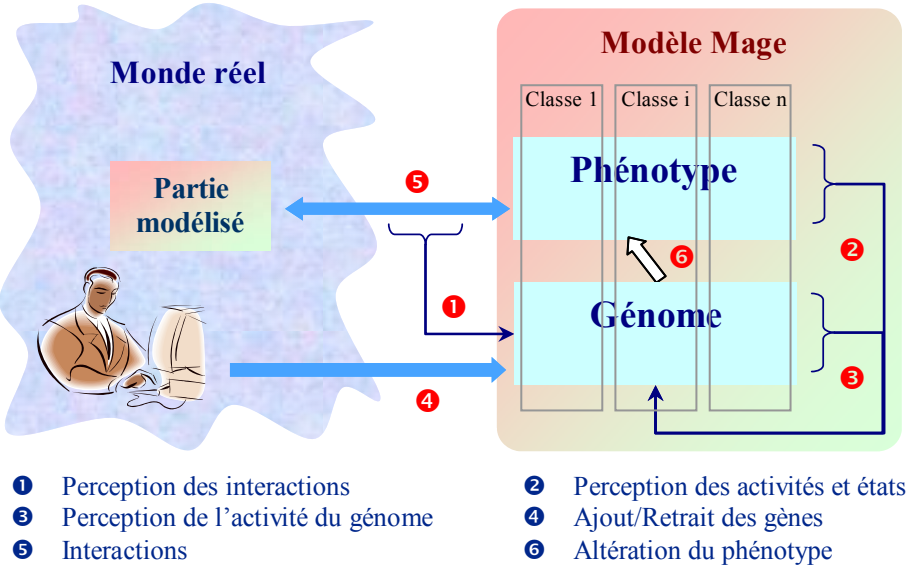


Figure 1.2. Les Concepts en Mage

1.5.4 Mage : Mode Opérationnel

Mage fait usage de quatre notions : le phénotype, le génome, l'interaction et les stimuli. On parle de phénotype à la fin de la phase d'embryogenèse. Le phénotype est décrit à l'aide d'un composant appelé composant universel. Le phénotype est ainsi un graphe constitué d'une multitude de composants universels simples ou composés et interconnectés pour former l'équivalent d'un système objet classique. Il consiste en des composants interconnectés appelés composants universels (UC). Le composant universel est une unité destinée à modéliser et abstraire tous les constituants d'un système : classe, méthode, objet primitif, objet d'une classe, tableau et instruction, graphe, sous graphe (Figure 1.3). Le phénotype est ainsi un graphe constitué d'une multitude de composants universels, simples ou composés, et interconnectés pour former l'équivalent d'un système objet classique.

Les composants universels ont une granularité variable et leur interconnexion fixe à la fois le flux de données et le flux de contrôle. Cette représentation du phénotype permet une expression plus simple et plus uniforme du génome. En effet, au lieu d'altérer un phénotype complexe, le génome altère des connecteurs, des imbrications de composants, des types de composants, etc.

Le génome est partagé en segments dits chromosomes. Chaque chromosome se compose d'éléments de faible granularité qui sont les gènes. On distingue trois types de chromosomes :

Chromosome D. Ce type regroupe les gènes constructeurs ou de développement, responsables des altérations des comportements et structures du phénotype.

Chromosome F. Ce type est composé des gènes qui garantissent des fonctionnalités d'ordre global comme le verrouillage et déverrouillage des composants lors du changement.

Chromosome C. Il matérialise le flux de contrôle de l'évolution et est constitué des gènes contrôleurs qui agissent par activation/désactivation sur tous les autres gènes.

Les gènes sont des objets dont la structure, immuable, se compose de quatre parties : type, état d'activation, condition de déclenchement et informations complémentaires.

La condition d'un gène lui permet de percevoir les stimuli. On en distingue 3 types :

Les stimuli factuels qui reflètent l'état d'un constituant du phénotype ou du génome (présence ou absence de composants universels, de gènes, de connexions, …).

Les stimuli d'activité qui indiquent qu'une activité du phénotype ou du génome est lancée ou terminée. Ils comprennent les stimuli d'interaction qui indiquent un échange entre le phénotype et le monde réel.

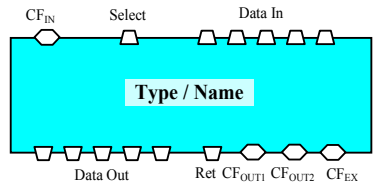
Les stimuli temporels qui permettent aux gènes de se déclencher indépendamment de la structure ou l'état du système, mais conformément à un repère temporel. Ce repère peut être celui d'une propriété, d'un objet ou d'une classe et peut être absolu (celui du système).

L'approche Mage utilise deux modèles d'exécution différents. Le premier est celui du phénotype et le second celui du génome. Au niveau du phénotype, le modèle est celui des langages orientés objets classiques. La sémantique associée à l'UC (composant universel) est la suivante :

Les UC qui matérialisent des données ou des classes n'utilisent pas les flux de contrôle. Il est possible d'y introduire une valeur par l'entrée Select et de lire celle stockée à partir de la sortie Ret.

Les UC qui matérialisent des méthodes ou des instructions utilisent le flux de contrôle et le flux de données. Le déclenchement de l'exécution de l'action associée à l'UC s'opère par l'arrivée d'un jeton sur CF_{IN} et la présence des données nécessaires sur les entrées d'information Select et $DI_0..DI_N$. Après traitement, le jeton est transmis sur CF_{OUT1} (CFOUT2 dans le cas de IF-Else si *Select* est à faux) et le résultat sur les sorties d'information $_{Ret}$ et $DO_0..DO_N$. Toute incompatibilité de données forcera l'UC à produire une exception qui

consiste à transmettre le jeton sur la sortie CFEX qui est reliée à un UC de type Method qui la traite.



Le composant universel (UC)

Type : Indique le rôle de l'UC (classe, méthode, instruction, …)
Name : Nom symbolique ou un nombre unique qui identifie l'UC. Si l'UC est une instruction d'appel, le nom indique l'objet dont la méthode est invoquée.
CF_{IN} : Entrée du flux de contrôle
CF_{OUT1} : Première sortie du flux de contrôle
CF_{OUT2} : Seconde sortie du flux de contrôle
CF_{EX} : Sortie du flux de contrôle dans le cas des exceptions
Select : Entrée non typée qui reçoit une valeur, une référence d'objet ou un sélecteur de méthodes
Ret : Sortie non typée donnant la valeur/référence d'objet ou le résultat d'une méthode
DataIn : DI_0 ..DI_N sont des entrées non typées utilisées pour recevoir des paramètres si l'UC est une méthode ou un appel, ou les indexes d'un élément si l'UC est un tableau
DataOut : Sortie de valeurs ou références d'objets résultat de l'exécution d'une méthode

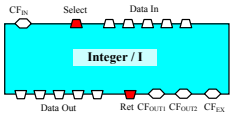
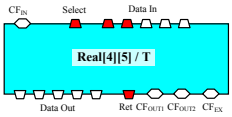
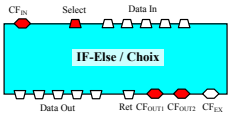
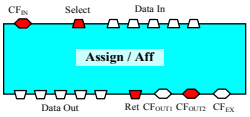
UC				
Ce qu'il représente	Un objet primitif de type Integer. Seuls *Select* et *Ret* sont utilisés. *Select* permet d'assigner une valeur à l'UC et *Ret* permet de la lire	Un tableau à deux dimensions de réels nommé *T*. *Select* et *Ret* permettent d'écrire/lire une valeur. DI_0 *et* DI_1 introduisent les indexes des valeurs.	Instruction *If-Else*. Le *Select* reçoit le *Ret* d'un sous-graphe évaluant une condition. Le flux de contrôle arrive par CF_{IN} puis est transmis sur CF_{OUT1} ou CF_{OUT2} selon la valeur du *Select*.	C'est une affectation qui établit une connexion entre deux objets. Le *Select* de l'objet assigné est connecté au *Ret* de *Aff* et la valeur assignée est donnée sur le *Select* de *Aff*.

Figure 1.3 Le composant universel

1.5.5 L'évolution Anticipée et Non Anticipée

Les systèmes ontogénétiques présentent la caractéristique d'évoluer dynamiquement et de manière autonome pour répondre aux besoins des utilisateurs et leurs changements qu'ils soient anticipés ou non, sachant que :

Les changements anticipés sont ceux identifiés avant la livraison des systèmes logiciels. Ils n'ont de signification que lorsque vient le moment opportun. Ces changements peuvent être anticipés au moment de la conception et doivent être prévus par le programmeur (concepteur) pendant les phases de développement du produit logiciel. Il serait aisé de les inclure moyennant un paramétrage dans le système. Par exemple, le passage d'un objet d'une classe à une autre (e.g. un candidat qui soutient une thèse devient enseignant et passe de la classe *Etudiant* à la classe *Enseignant* [Ous 99].

Les changements non anticipés sont définis dans [Alm 06] comme : "unanticipated software evolution is not something for which we can prepare during the design of a software system". Que l'on peut traduire par "Les évolutions non anticipées ne peuvent être prises en compte durant la conception des systèmes logiciels". Ils visent à corriger les erreurs, ajouter de nouvelles fonctionnalités et en supprimer d'autres. Ces dernières apparaissent une fois le système opérationnel. Ce sont des changements que l'équipe de développement du produit n'a pas considéré dans le processus de conception et d'implémentation.

Les évolutions non anticipées sont une voie de recherche clé en génie logiciel, car elles sont fortement liées au temps et aux coûts du développement et maintenance des systèmes logiciels. Cette dernière est une activité très couteuse et demande pour sa réalisation beaucoup d'effort et de temps pour permettre aux mainteneurs de comprendre les programmes.

Malgré l'importance de l'évolution des systèmes logiciel, les modèles et techniques qui offrent un support pour l'évolution des logiciels sont loin d'être idéales. En particulier, les changements non anticipés des besoins qui ne sont pas bien pris en charge, malgré qu'ils représentent des complications techniques et des coûts très importants. L'objectif des travaux de recherche est alors de trouver des méthodes pour que l'évolution non anticipée se fasse au moindre coût. Dans la littérature, la plupart des propositions reposent sur la transformation de code [Sad 03].

Conclusion

Le paradigme Mage présente une nouvelle approche où le modèle d'un système est une représentation qui englobe ses dimensions structurelle, comportementale et ontogénétique. Mage incarne une vision radicale du changement en s'inspirant du développement biologique. En effet, sous Mage, un système est conçu pour changer. Pour forcer cette vision, le phénotype est considéré inexistant au départ et qu'il est le fruit d'une activité continue du génome. Mage modélise uniformément le phénotype en utilisant le composant universel et le génome par trois types de gènes dont les rôles sont bien distincts. En plus de son aspect naturel, Mage traite de façon uniforme les changements anticipés et non anticipés. Les changements anticipés sont prévus, donc déjà étudiée lors de la phase d'analyse alors que ceux non anticipés, sont imprévus lors du développement du système.

La prise en compte de l'ontogenèse constitue un nouveau défi et une vision radicale de l'évolution qui a un effet aussi bien sur notre façon de percevoir les systèmes logiciels que notre manière de les concevoir. En effet, les systèmes ontogénétiques nécessitent clairement des démarches de développement appropriées qui redonnent à l'évolution le statut de concept central. Dans le chapitre suivant nous discutons les deux approches de méthodologies de développement existantes et nous présentons le processus Rational Unified Process basé sur le formalisme UML.

CHAPITRE 2
LES DEMARCHES DE DEVELOPPEMENT ACTUELLES

Afin de faire face à la complexité croissante du développement logiciel, des méthodes de gestion des projets informatiques ont été introduites. Celles dites traditionnelles sont axées-plan et généralement lourdes, ces méthodes ont perdu de plus en plus de partisans en faveur des nouvelles méthodes dites « agiles ». Cependant, ces dernières présentent des limitations pour les projets critiques ou de grande taille nécessitant une équipe importante en nombre.

Ce chapitre vise à faire une présentation succincte permettant de comprendre les processus et les méthodes de développement existants tout en comparant l'école traditionnelle et celle dite Agile. Nous présentons également le processus RUP qui constitue le cadre de la démarche que nous proposons dans le chapitre 4.

2.1 Le Processus de Développement

La recherche en génie logiciel a depuis long temps tenté de mieux comprendre le processus de développement logiciel, minimalement, pour en reproduire les bonnes pratiques, et idéalement pour pouvoir le mécaniser. Un processus est «un ensemble organisé d'activités mises en œuvre dans un but précis».

Le développement de logiciel comprend l'ensemble des étapes et processus qui permettent de passer de l'expression d'un besoin informatique à un logiciel fonctionnel et fiable. L'application de différentes activités allant de la collecte et l'analyse des besoins jusqu'au déploiement et la maintenance du système.

Le processus de développement de systèmes logiciels est alors défini comme étant un ensemble organisé d'activités mises en œuvre dans le but de construire un logiciel. Plus précisément, un processus de développement de logiciels est définis comme «un ou plusieurs ensembles d'activités ordonnées, avec un déroulement séquentiel ou parallèle, qui permettent d'analyser et de concevoir, de rétro-concevoir ou de refondre le système» [Cas 92].

2.1.1 Définitions et Concepts

Système. Un système est généralement décrit comme un ensemble d'éléments en interaction organisés en un tout homogène au sein d'un environnement avec lequel il interagit [Moi 90]. Il évolue dans le temps [Sha 67], il transforme des grandeurs d'entrées en grandeurs de sorties [Lar 93] et il réalise des fonctions afin d'atteindre un objectif donné (Figure 2.1).

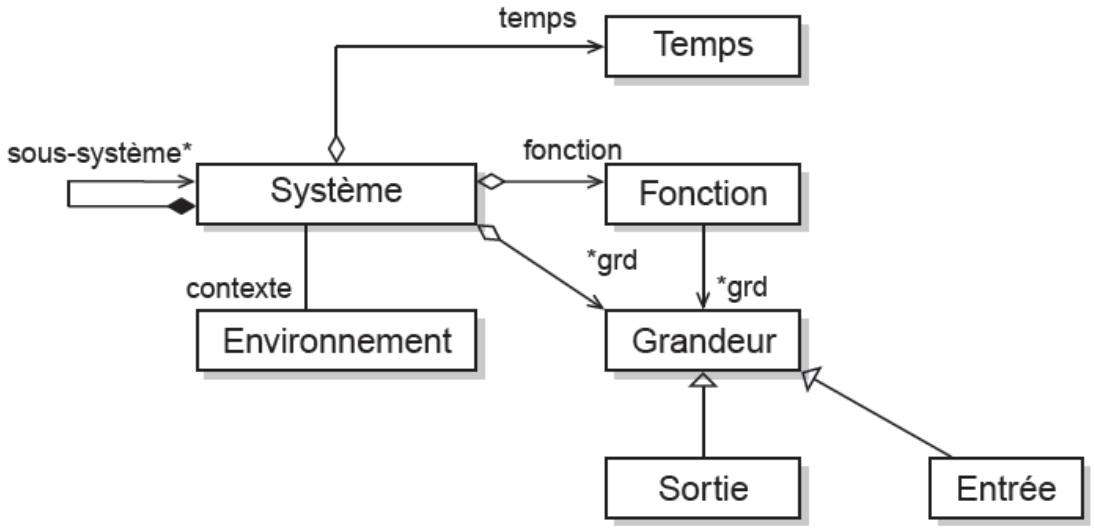


Figure 2.1 Vue conceptuelle de la notion de système

Système logiciel. Les systèmes logiciels, comme tout système, sont constitués de sous systèmes en interactions et évoluent dans un environnement donné. A cette caractérisation, se greffe un certain nombre de concepts clefs tels que : *l'architecture, l'état, le comportement, la réactivité, la concurrence, la communication, l'émergence ou encore la complexité* [Per 07].

Méthode. Pour Booch [Boo 91], une méthode est « un processus rigoureux permettant de générer un ensemble de modèles qui décrivent divers aspects d'un logiciel en cours de construction en utilisant une certaine notation bien définie ».

Les modèles de processus proposent des démarches de développement de systèmes logiciels. Ils décrivent une démarche méthodologique pour atteindre la cible que sont les produits [Rol 05].

Humphrey définit le processus d'ingénierie logicielle comme suit : «Le processus d'ingénierie logicielle est l'ensemble des activités d'ingénierie logicielle nécessaires à la transformation des besoins d'un utilisateur en un logiciel» [Hum 88].

2.1.2 Cycle de vie de logiciel

Le cycle de vie d'un logiciel est un ordonnancement de différentes étapes du processus de développement. Il désigne toutes les étapes de développement d'un logiciel, de sa conception à sa disparition. L'objectif d'un tel découpage est de permettre de définir des jalons intermédiaires permettant la validation du développement logiciel, c'est-à-dire la conformité du logiciel avec les besoins exprimés, et la vérification du processus de développement, c'est-à-dire l'adéquation des méthodes mises en œuvre. L'origine de ce découpage provient du constat que les erreurs ont un coût d'autant plus élevé qu'elles sont détectées tardivement dans le processus de réalisation. Le cycle de vie permet de détecter les erreurs au plus tôt et ainsi de maîtriser la qualité du logiciel, les délais de sa réalisation et les coûts associés.

Le cycle de vie du logiciel comprend au minimum les étapes suivantes (Figure 2.2):

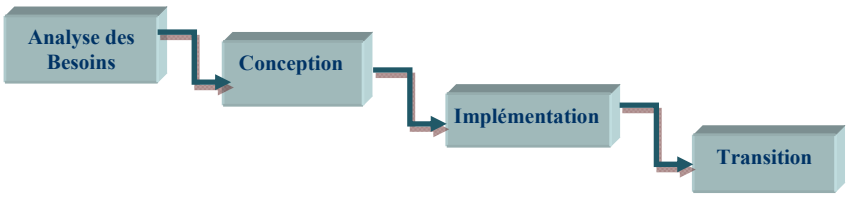


Figure 2.2 Etapes essentielles d'un cycle de vie de logiciel

2.1.2.1 Le cycle de développement en cascade

Le modèle de cycle de vie en cascade a été mis au point dès 1966, puis formalisé aux alentours de 1970. Il définit des phases séquentielles à l'issue de chacune desquelles des documents sont produits pour en vérifier la conformité avant de passer à la suivante (Figure 2.3).

Le cycle en « cascade » se caractérise par des phases séquentielles, qui le succèdent après la validation des livrables produits lors de la phase précédente :

- Tous les besoins sont exprimés et recueillis lors de la première phase, puis vient l'analyse détaillé de ces besoins dont la conception du système est très dépendante.
- La conception du système, bien que textuelle ou représentée sous forme de diagrammes, doit être validée avant le démarrage du développement.

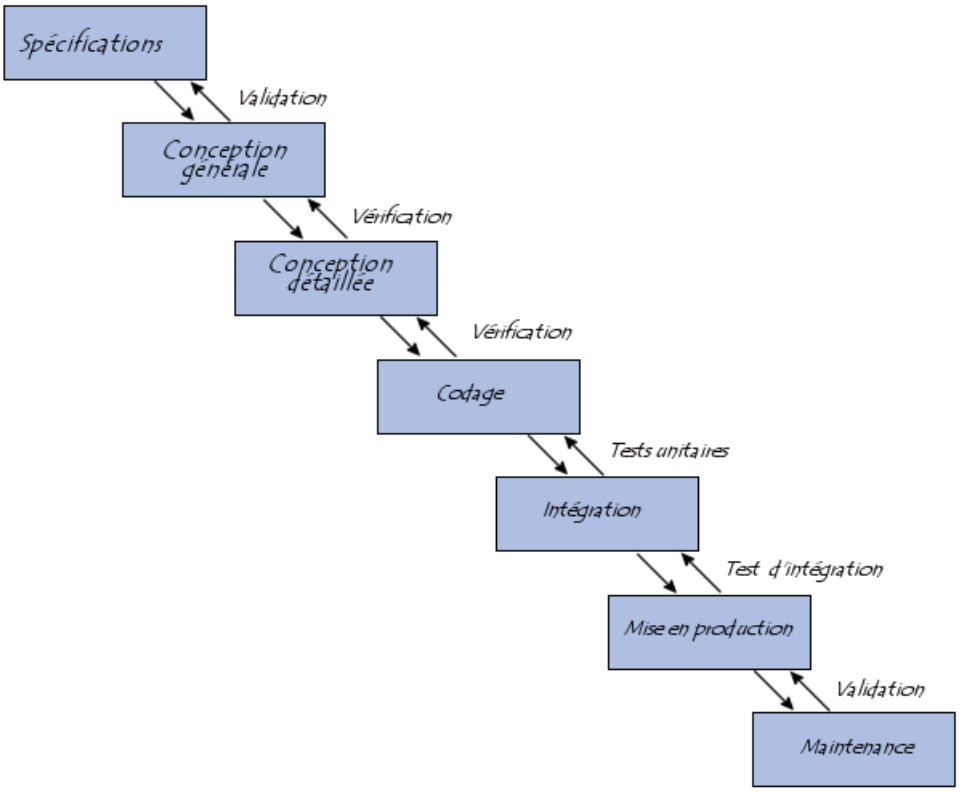


Figure 2.3 Cycle de vie de logiciels en cascade

- Le développement doit être achevé pour permettre à l'équipe de testeurs de lancer ses campagnes de tests fonctionnels et techniques.
- Enfin, une fois les anomalies ont été corrigées, on peut procéder à l'intégration globale finale et à la mise en production du système.

2.1.2.2 Le modèle de cycle de vie en V

Le modèle du cycle en V a été imaginé pour pallier au problème de réactivité du modèle en cascade. Ce modèle est une amélioration du modèle en cascade qui permet en cas d'anomalie, de limiter un retour aux étapes précédentes. Les phases de la partie montante ci-contre doivent renvoyer de l'information sur les phases en vis-à-vis lorsque des défauts sont détectés afin d'améliorer le logiciel. Le cycle en V est devenu un standard de l'industrie du développement de logiciel et de la gestion de projet depuis les années 1980 [McD 84].

Le cycle en V améliore le cycle de la cascade pour mettre l'accent sur les aspects vérification et validation. Les jeux de tests sont préparés dès les phases de spécification et permettent ainsi d'améliorer la production de logiciels corrects et valides (norme AFNOR Z67-130) [Gir 07].

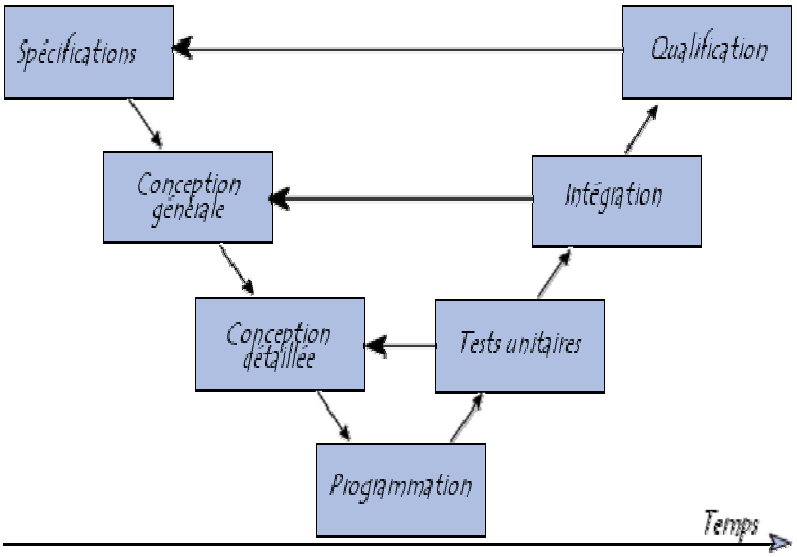


Figure 2.4 cycle de vie en V [Boe 81]

L'inconvénient évident du modèle en V est qu'il faut attendre la fin du processus de développement pour avoir le retour des utilisateurs sur le système logiciel réellement implanté.

2.1.2.3 Le modèle de cycle de vie en Spirale

Le développement se base sur les différentes étapes du cycle en V pour permettre de produire un produit de plus en plus complet via la réalisation de versions successives. Le cycle en spirale généralise le principe d'incrément et évite « l'effet tunnel » où le temps est trop long entre les premières spécifications de besoins et la livraison de tout le système [Boe 86]. Il introduit la notion de prototype (Figure 2.5). Ce cycle de vie correspond à une adaptation du principe de la roue de Deming [Gir 07] (Figure 2.6). La roue de Deming a été créée par Willian Edwards Deming, aussi connue sous le nom de méthode PDCA. La PDCA tire son origine des premières lettres des mots qui la composent : Plan-Do-Check-Act. Ces derniers peuvent être interprétés tel qu'il suit :

- Plan : Préparer, Planifier ;
- Do : Développer, réaliser, mettre en œuvre ;
- Check : Contrôler, vérifier ;
- Act (ou Adjust): Agir, ajuster, réagir.

La méthode PDCA est une démarche cyclique d'amélioration qui consiste, à la fin de chaque cycle, à remettre en question toutes les actions précédemment menées afin de les améliorer [PCD 14].

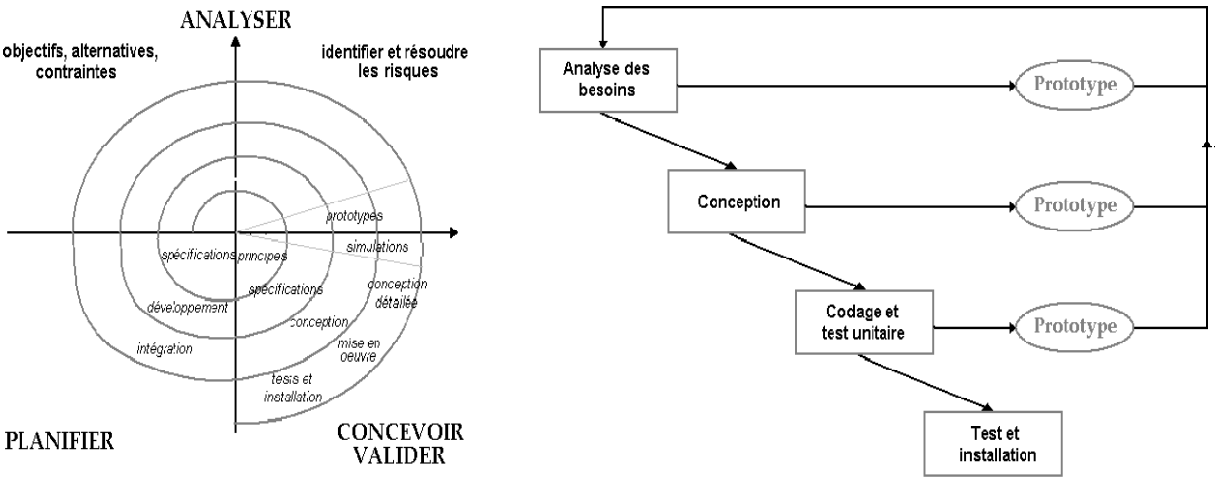


Figure 2.5 Modèle en Spirale de Bohem

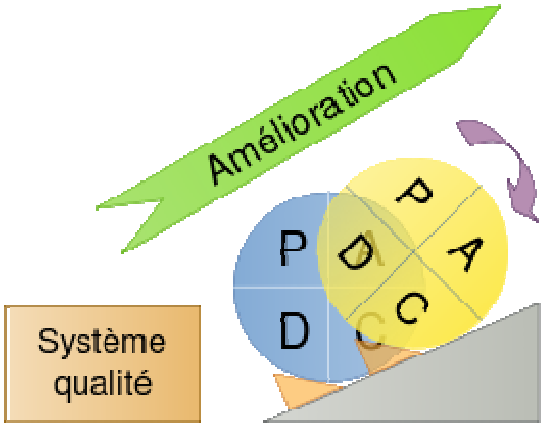


Figure 2.6 Roue de Deming

2.1.3 Contexte du développement d'un système logiciel

Il est important de mieux situer le contexte général de tout développement d'un système logiciel. Un système logiciel est le résultat d'un processus d'ingénierie mettant en œuvre les technologies courantes pour satisfaire les exigences spécifiques d'un client.

Trois aspects essentiels (Figure 2.7), correspondant globalement à trois métiers différents, sont à considérer [Col 10] :

1. **le *métier***, qui requiert des compétences liées au domaine d'application du logiciel,
2. **l'*ingénierie logicielle***, qui requiert des compétences en conception de logiciels,

3. **la *gestion du processus*** *de développement*, qui requiert des compétences en génie logiciel.

Le métier ou domaine d'application impose la prise en compte de connaissances métiers ou de connaissances théoriques relevant d'un domaine d'application particulier. Il détermine des classes d'applications et peut avoir un impact important sur l'aspect spécificité d'un processus de développement (intégration d'activités ou d'outillage spécifiques, conception ou utilisation d'un Framework métier, etc.).

L'ingénierie logicielle regroupe l'ensemble des techniques, technologies, solutions logicielles ou encore méthodes de développement à utiliser pour concevoir le logiciel. Elle a un impact sur la spécificité du processus de développement (choix d'une approche orientée objets, ou à base de composants ou d'agents, conformité à un style d'architecture, intégration de Framework techniques, etc.).

La gestion du processus impose un cycle de développement ou une démarche méthodologique qui doivent être suivis par l'équipe de développement.

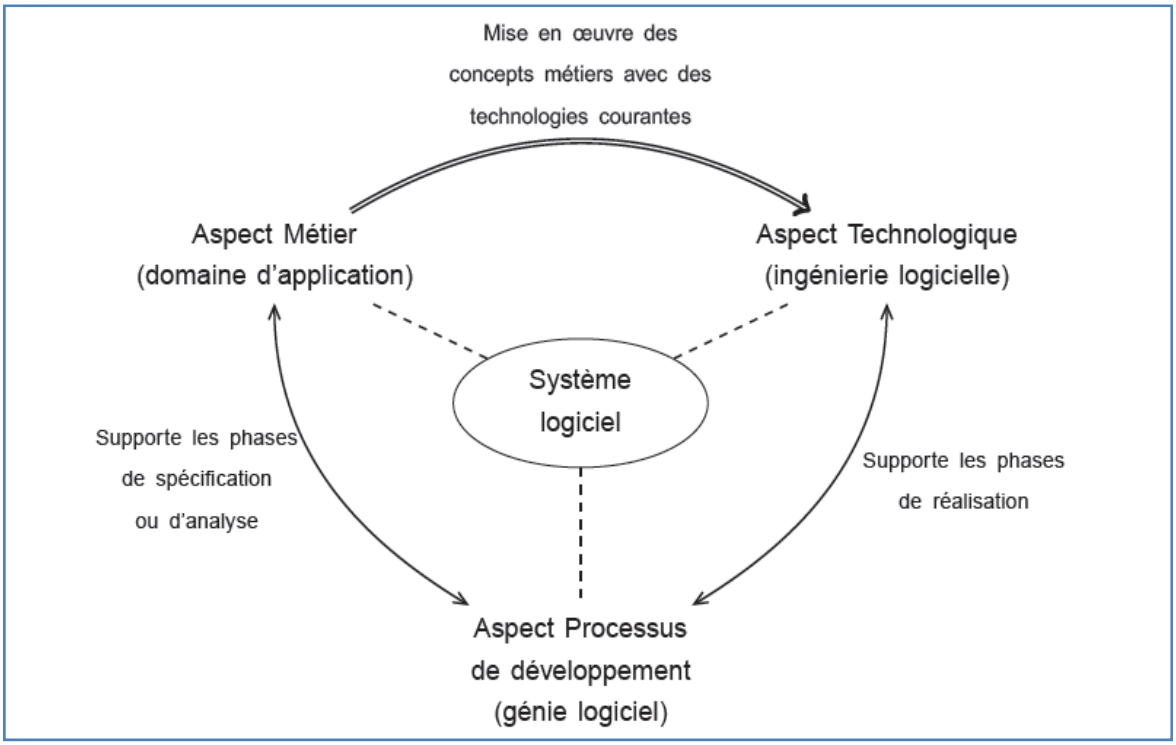


Figure 2.7 Contexte général du développement logiciel [Col 10]

Ces trois aspects sont en interaction et visent à répondre aux questions suivantes concernant le système en développement : *pourquoi* et *quoi* pour l'aspect métier, *avec quoi* pour l'aspect ingénierie logicielle et enfin *comment faire* pour l'aspect processus. Un processus spécifique à une classe d'applications relevant d'un domaine métier particulier devra alors intégrer ces différents aspects.

2.2 Réflexion sur les Approche de Développement des Systèmes Logiciels

Depuis environ deux décennies, deux types d'approches de développement logiciel se démarquent : celles dites traditionnelles [Ket 05, Ner 05], ou basée sur l'ingénierie [Fow 05, Ger 05] et les autres dites agiles [Coc 02, Fow 05] ou légère [Zet 01, Naw 02, Mes 07, Kru 03]. Nous décrivons leurs principes dans ce qui suit.

2.2.1 Les Approches Traditionnelles

Les activités typiques d'un cycle de vie sont la définition des besoins, la conception, le développement, l'implantation et enfin les tests avant la livraison du produit au client. Une première façon d'organiser ces activités conduit aux cycles de vie dits prédictifs dont des illustrations sont le cycle en *V* ou le cycle en *Cascade* [Roy 70]. Ces cycles sont dits *prédictifs* car ils demandent, dès l'identification et la mise en place du cycle, de définir toutes les échéances qui en jalonneront l'évolution. Ces cycles ont fait incontestablement la preuve de leur efficacité dans des développements de produits bien définis grâce à la rigueur qu'ils imposent. Cependant, aujourd'hui, ils ont montré des limites dans certains secteurs (développement de produits innovants par exemple) où une maîtrise totale de la planification n'est pas possible. Dans certains domaines d'application, ces cycles ne se sont pas révélés comme étant réalistes face au déroulement réel des processus de développement [Som 06].

2.2.1.1 Origine des Approches Traditionnelles et du Génie Logiciel

L'industrie logicielle a intégré la société moderne depuis plus de 50 ans. À la fin des années 50, l'informatique est devenue de plus en plus populaire et s'est étendue à d'autres disciplines. Cet élargissement à un public non scientifique, a engendré la création du métier de programmeur. L'utilisateur exprimait ses besoins et le programmeur réalisait la solution informatique. Ce fût le premier écart qui sépara les professionnels de l'informatique et les utilisateurs. À la fin des années 60, les grands systèmes commerciaux ont démontré qu'il était difficile d'adapter à grande échelle, les principes jusque là convenables. Les grands projets dépassaient les budgets et les échéanciers, ce fût alors la crise du génie logiciel [Tre 07]. Le développement logiciel n'était alors que le résultat d'activités désordonnées dites « code and fix » sans planification, ni conception détaillée en amont [Ben 12].

Depuis des décennies, les projets sont gérés avec une approche classique, le plus fréquemment « en cascade » ou son adaptation « en V », basée sur des activités séquentielles : on recueille les besoins, on définit le produit, on le développe puis on le teste avant de le livrer au client. Ces méthodologies se caractérisent par un attachement considérable à essayer de tout planifier, « tout doit être prévisible », au tout début du projet. C'est pourquoi on les qualifie d'approches « prédictives ». Un plan de management du projet décrit comment et quand le travail sera réalisé, les modalités de planification, d'exécution, de suivi et de clôture du projet.

Cette volonté persistante de vouloir piloter le projet par les plans (*plan-driven development*) [Ben 12, Mes 08], a conduit les acteurs des projets à redouter, voire à s'opposer systématiquement à tout changement : changement dans le contenu ou le périmètre du projet, dans le processus de développement, au sein de l'équipe, bref à toute modification des plans initiaux, auxquels on doit rester conforme. Les méthodes axées-plan sont considérées comme étant le moyen traditionnel pour le développement logiciel. Fondées sur des concepts tirés des principaux domaines de l'ingénierie, ces méthodes perçoivent le développement selon un paradigme de besoins/conception/développement couplé de processus standards et bien définis que l'organisation améliore constamment.

2.2.1.2 Limites des approches traditionnelles

Nous décrivons ci-après les limites des approches traditionnelles ou ce qui fait leurs points faibles.

La rigidité de l'approche. On déplore que la nouveauté, la marge de manœuvre laissée au client pour préciser ou faire évoluer ses attentes, la non-prévisibilité de tous les événements soient difficilement compatibles avec une approche prédictive comme celle du cycle en cascade.

L'approche « en cascade » est par conséquent trop rigide pour permettre des retours en arrière; elle suppose que l'on fasse bien du premier coup. Une décision ou une anomalie détectée dans une phase aval de la cascade peuvent remettre en cause partiellement ou totalement des travaux validés précédemment et considérés comme définitifs.

L'effet tunnel. L'effet tunnel est une autre des caractéristiques de l'approche «en cascade» : Si un projet dure un an et la phase de recueil des besoins dure deux mois ; le client ne voit le résultat que neuf mois plus tard (Figure 2.8)

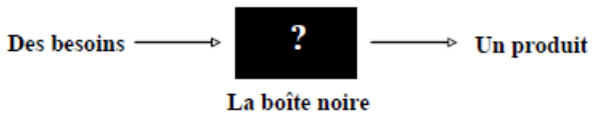


Figure 2.8 La « boîte noire »

D'une part, la non-transparence des équipes de développement suscite des problèmes quant à leur capacité à coopérer, d'autre part, la longueur des phases techniques auxquelles le client n'est pas associé rend celui-ci dubitatif sur le résultat à venir. Ce qui ne favorise pas la collaboration efficace entre informaticiens et utilisateurs. D'autant plus si le résultat livré n'est pas conforme à ce qui est attendu.

Une mauvaise communication. L'absence de jalons intermédiaires prohibe la validation de ce que sera la version finale du produit. Il faut attendre que la phase de développement soit bien avancée pour découvrir les premiers écrans. Les mauvaises surprises en fin de cycle de vie et le refus du changement par les équipes de développement pénalisent la qualité et les relations avec les utilisateurs. Elles en deviennent parfois même conflictuelles ; les uns s'attachent fermement à leurs plans initiaux pour livrer ce qui était convenu à l'échéance prévue, même si le résultat ne correspond plus ou pas totalement à ce qui est réellement attendu; les autres ressentent cette rigidité comme un désintérêt pour la valeur ajoutée du produit final.

La succession d'intervenants, au travers des différents corps de métier, nuit également à la fluidité de l'information, crée même une déperdition d'information et d'énergie ainsi que de nombreuses ruptures de charge [Mes 08].

La levée tardive des facteurs à risques. Dans un cycle de vie « en cascade », les facteurs à risques sont levés tardivement, puisque les tests de performance ou d'intégration, par exemple, sont reportés après les développements, tout comme l'appréciation des interfaces homme-machines qui sont souvent sujettes à des débats subjectifs et longs.

L'impact des risques augmente avec l'avancement du projet, puisque plus une anomalie est détectée tardivement, plus le retour arrière est complexe, plus sa correction coûtera cher et plus les effets de bords seront importants.

Une documentation disparate. Afin de se prémunir contre ces risques, l'approche « en cascade » s'attache fortement à la production d'une documentation importante.

La documentation permet de repousser le moment où il va falloir aborder la phase de codage, phase irréversible.

Elle rassure et apporte la preuve que la réalisation progresse; elle matérialise l'avancement et engage les parties prenantes. En effet, il est plus aisé de s'opposer au changement en brandissant un document contractuel validé précédemment.

Malheureusement, cette documentation, souvent trop exagérée, ne reflète pas la réalité des développements : on a beau valider un document d'architecture, celle-ci reste théorique et conceptuelle tant qu'elle n'est pas implémentée et testée dans des conditions réelles; on a beau présenter des maquettes papier au client, celui-ci est plus sensible à ce qu'il voit concrètement sur un écran.

Au final, on s'interroge sur l'utilité de cette documentation, qui n'est, en outre, pas toujours mise à jour tout au long du projet et devient donc vite inexploitable.

2.2.2 Approche de Développement Agile

Les méthodes agiles se sont développées séparément, jusqu'en 2001, où leurs créateurs sont arrivés à un consensus sous le label «Agile». Les traits essentiels et communs ont alors été formalisés en un manifeste nommé «le manifeste agile» (Manifeste Agile, 2001) prônant quatre valeurs principales:

1. Priorité aux personnes et aux interactions par rapport aux procédures et outils,
2. Priorité aux applications fonctionnelles par rapport à la documentation exhaustive,
3. Priorité de la collaboration avec le client par rapport à la négociation de contrat,
4. Priorité de l'acceptation du changement par rapport à la planification.

Ces quatre valeurs se déclinent en principes au nombre de treize principes. Nous les présentons avec les explications dans la table 2.1.

Principes	Description
La priorité est de satisfaire le client en lui livrant très tôt et régulièrement des versions opérationnelles de l'application, source de valeur.	Grâce au développement itératif, chaque livraison intermédiaire donne lieu à une validation par le client ; son feedback est essentiel pour garantir la conformité de la livraison avec ses attentes, pour prendre en compte ses remarques et (re)prioriser.
Accepter le changement dans les exigences, même tard dans le cycle de vie, pour garantir la compétitivité du client.	Cet état d'esprit caractérise une équipe agile qui démontre ainsi sa capacité à comprendre et apprendre comment satisfaire encore mieux la demande.
Livrer le plus souvent possible des versions opérationnelles de l'application, à une fréquence allant de deux semaines à deux mois.	Une version intermédiaire du produit final, visible et testable, satisfait davantage le client qu'une documentation à valider. Il a, de ce fait, la preuve tangible que le projet avance et peut exprimer son point de vue sur le résultat présenté.
Client et développeurs doivent coopérer quotidiennement tout au long du projet.	Les relations conflictuelles ne font pas partie de l'esprit agile ; on préférera des relations collaboratives et de partenariat basées sur la confiance et le consensus. Le client (ou son représentant) est accessible et disponible, totalement impliqué dans toutes les phases du projet.
Construire des projets autour d'individus motivés. Leur donner l'environnement et le support dont ils ont besoin et leur faire confiance pour remplir leur mission.	Le facteur clé du succès d'un projet est l'équipe. Tout obstacle à son bon fonctionnement devra être levé ; un changement, s'il s'avère nécessaire, sera apporté aux processus, aux outils, à l'environnement, à la composition de l'équipe
La méthode la plus efficace de communiquer des informations à une équipe et au sein de celle-ci reste la conversation en face à face.	Par défaut, on privilégie l'échange oral à l'écrit, pour lever toute ambiguïté et favoriser la rapidité de la compréhension. Tout ne peut être formalisé par écrit, notamment la « connaissance tacite », c'est-à-dire 'information « informelle », la culture du projet, détenues par chacun au sein d'une équipe.

Le fonctionnement de l'application est le premier indicateur d'avancement du projet.	Il n'existe pas d'autre indicateur plus pertinent que le pourcentage ou le nombre d'exigences satisfaites ; on ne mesure pas un projet à la quantité de documents produits ou au nombre de lignes de code, non significatifs pour le client.
Les méthodes agiles recommandent que le projet avance à un rythme soutenable.	La qualité du travail fourni dépend du rythme de travail qui doit être adapté en fonction des spécificités du projet. Le rythme doit être soutenu et soutenable sur la durée du projet.
Sponsors, développeurs et utilisateurs devraient pouvoir maintenir un rythme constant indéfiniment.	Ce rythme de travail est à déterminer par l'ensemble des membres de l'équipe et par le client, en fonction de la productivité de l'équipe et des priorités du client. Mais travailler en heures supplémentaires, surtout pour corriger des bogues ou des régressions, n'apporte aucune valeur ajoutée.
Porter une attention continue à l'excellence technique et à la conception améliore l'agilité.	Maintenir un code propre, évolutif et performant est un objectif permanent de l'équipe ; il ne s'agit pas de produire rapidement du code non exploitable par les autres, ni du « jetable ». De plus, cela évite surtout d'enliser les développements ultérieurs, avec des modifications cassant un développement fragile, nécessitant des interventions à des endroits variés du code.
La simplicité – L'art de maximiser la quantité de travail à ne pas faire – est essentielle.	La simplicité garantit l'évolutivité du système. La complexité, au contraire, coûte davantage et rend plus difficiles les évolutions inhérentes au développement incrémental. La conception ne doit comporter que des éléments utiles.
Les meilleures architectures, spécifications et conceptions sont le fruit d'équipes qui s'auto-organisent.	Le chef de projet agile n'est plus celui qui attribue des tâches. L'équipe, elle-même, se responsabilise et définit ses travaux à réaliser, le partage des tâches s'effectuant sur la base du volontariat.
À intervalles réguliers, l'ensemble de l'équipe s'interroge sur la manière de devenir encore plus efficace, puis ajuste son comportement en conséquence.	L'environnement d'un projet n'est pas constant ; l'équipe agile, qui en a conscience, s'interroge en permanence sur la façon d'améliorer son fonctionnement afin de s'adapter aux nouvelles conditions. C'est aussi l'acceptation du changement.

Table 2.1 : Les treize principes du manifeste Agile [Mes 08]

Les treize principes fondamentaux Agiles étaient déclarés être :

- La première priorité est de satisfaire le client en livrant tôt et régulièrement des logiciels utiles.
- Le changement est accepté, même tardivement dans le développement. Les processus agiles exploitent le changement comme avantage compétitif pour le client.
- Livrer fréquemment une application fonctionnelle, toutes les deux semaines à deux mois, avec une tendance pour la période la plus courte.
- Les gens de l'art et les développeurs doivent collaborer quotidiennement au projet.
- Bâtir le projet autour de personnes motivées. Donner-leur l'environnement et le soutien dont elles ont besoin, et croyez en leur capacité à faire le travail.

- La méthode la plus efficace de transmettre l'information est une conversation en face à face.
- Un logiciel fonctionnel est la meilleure unité de mesure de la progression du projet.
- Les processus agiles préconisent un rythme de développement soutenable. Commanditaires, développeurs et utilisateurs devraient pouvoir maintenir le rythme indéfiniment.
- Une attention continue à l'excellence technique et à la qualité de la conception améliore l'agilité.
- La simplicité - l'art de maximiser la quantité de travail à ne pas faire - est essentielle.
- Les meilleures architectures, spécifications et conceptions sont issues d'équipes qui s'auto-organisent.
- À intervalle régulier, l'équipe réfléchit aux moyens de devenir plus efficace, puis accorde et ajuste son comportement dans ce sens.

2.2.3 Comparaison entre l'Approche de Développement Agile et Traditionnelle

Les méthodes traditionnelles et les méthodes agiles se sont développées dans deux milieux diamétralement opposés. Les premières sont venues introduire de la discipline dans les grands projets gouvernementaux ayant des coûts d'échec colossaux. Les méthodes agiles, quant à elles, se sont développées dans la « bulle » du Dotcom sur des petits projets web avec des coûts d'échec frôlant le zéro. Les méthodes agiles ont introduit un nouveau paradigme pour pallier les défaillances des méthodes traditionnelles et garantir un meilleur taux de succès des projets.

Cependant, des études empiriques ont démontré que le paradigme agile n'est pas approprié à toutes les situations et que les approches traditionnelles offrent dans certains cas de meilleurs résultats. En effet, l'une des plus grandes limitations des méthodes agiles est la grande taille des équipes. Cockburn et HighSmith [Coc 01a] concluent qu'avec l'augmentation de la taille, la coordination entre les interfaces devient un problème crucial. Les auteurs dans [Coc 01a] et [Fow 05] indiquent que la communication face-à-face prônée par les méthodes agiles devient difficile et complexe si l'équipe de développement dépasse les 20 personnes. En revanche, les méthodes traditionnelles axées-plan sont mieux appropriées aux projets à grande échelle.

Une autre limitation considérable des méthodes agiles consiste en la gestion des projets critiques. Les mécanismes de contrôle de qualité incorporés dans les méthodes agiles ne

prouvent pas que les produits développés soient sans danger. Les spécifications formelles, les tests rigoureux et d'autres techniques d'analyse et d'évaluation inclus dans les méthodes traditionnelles offrent de meilleurs mécanismes, même s'ils sont plus lourds, pour gérer les projets critiques [Ben 12]. La table 2.2 résume les principales différences entre ces deux approches.

Critères	Méthodes traditionnelles	Méthodes agiles
Approche	Prédictive	Adaptative
Mesure du succès	Conformité au plan	Valeur client
Domaine	Prévisible	Imprévisible/exploratoire
Retour sur investissement	A la fin du projet	Au début du projet
Style de management	Autocratique	Décentralisé
Culture	Commande, contrôle	*Leadership*, collaboration
Environnement	Stable, faible taux de changement	Turbulent, taux élevé de changement
Relation client	Interaction au besoin, insistance sur les exigences du contrat	Un représentant du client est dévoué au projet, insistance sur la priorisation des incréments
Perspective de changement	Durabilité de changement	Adaptabilité au changement
Focus	Processus	Humain
Communication	Explicite	Tacite
Documentation	Lourde	Faible
Cycles	En nombre limité	Nombreux
Planification en amont	Complète	Minimale
Taille du projet	Grande taille	Petite taille
Taille de l'équipe	Grande taille	Petite taille/créative
Développement	Conception extensive, longs incréments	Conception simple, courts incréments

Table 2.2 Différences entre les méthodes agiles et traditionnelles [Ben 12]

Des études empiriques [Coc 01, Con 01, Fow 01] ont démontré que le paradigme agile n'est pas approprié à toutes les situations et que les approches traditionnelles offrent dans certains cas de meilleurs résultats. En effet, l'une des plus grandes limitations des

méthodes agiles est la grande taille des équipes. Cockburn et HighSmith [Coc 01a] concluent qu'avec l'augmentation de la taille, la coordination entre les interfaces devient un problème crucial. L. Constantine [Con 01] et M. Fowler [Fow 01] indiquent que la communication face-à-face prônée par les méthodes agiles devient difficile et complexe si l'équipe de développement dépasse les 20 personnes. En revanche, les méthodes traditionnelles axées-plan sont mieux appropriées aux projets à grande échelle.

La communication dans l'approche Agile est tacite [Ben 12]. Les malentendus, incompréhensions, incohérences sont mis en évidence tôt dans le projet, il est donc encore possible de les corriger [Mes 08]. L'utilisateur a la possibilité de clarifier ses exigences au fur et à mesure. Le client reçoit des « preuves » tangibles de l'avancement du projet. À chaque itération, on sélectionne, avec le client, les fonctionnalités qui seront détaillées puis développées, en fonction de leur priorité et on établit le microplanning correspondant aux activités nécessaires pour le développement de ces fonctionnalités.

2.3. Rational Unified Process et UML

2.3.1 Unified Modeling Language (UML)

Dans le secteur du développement logiciel, suite à l'adoption massive de l'approche objet pour la réalisation des applications, et face aux besoins des développeurs d'avoir à leur disposition de nouvelles méthodologies d'aide à la conception objet, une multitude de méthodes a été lancée dont une cinquantaine entre 1990 et 1995, chacune essayant de s'imposer sur le vaste marché du développement logiciel [Col 10]. En 1995, cependant, répondant à un appel à projet lancé par l'**OMG** (*Object Management Group*) pour l'obtention d'une méthodologie standard de modélisation, trois spécialistes recrutés par la société **Rational Software** et ayant chacun proposé une méthode particulière, décident de spécifier un nouveau langage. Comprenant l'impossibilité de s'orienter vers une méthode unique, mais désireux d'obtenir un langage standardisé, ils créèrent **UML** (*Unified Modeling Language*) [Mul 00]. Né de la fusion des méthodes objet dominantes (OMT, Booch et OOSE), puis normalisé par l'OMG en 1997, UML est rapidement devenu un standard incontournable [Col 10]. UML n'est pas à l'origine des concepts objet, mais il en donne une définition plus formelle et apporte la dimension méthodologique qui faisait défaut à l'approche objet (Figure 2.10).

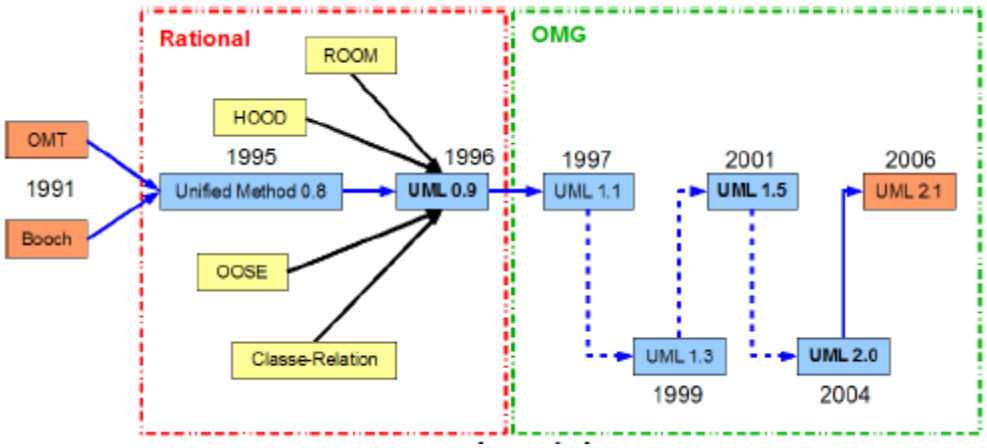


Figure 2.10 Ligne de vie du langage UML

Dans les spécifications du langage on distingue deux grandes catégories de description pour un système donné, on retrouve ces deux visions dans certaines méthodes qui ont inspiré les premières versions du langage :

– Une ***description structurelle***, permettant la représentation des éléments d'un système de façon structurée et en offrant une vision statique.

– Une ***description comportementale***, dans laquelle on retrouve les spécifications dynamiques à la fois internes aux objets identifiés et décrits dans la partie structurelle mais également les interactions entre ces différents objets.

Aucun guide d'utilisation n'est fourni avec l'ensemble des diagrammes dont l'utilisateur peut se servir. C'est l'une des principales difficultés pour le concepteur qui doit choisir de façon pertinente la bonne approche qui convient au cadre de son projet, mais c'est aussi pour le langage un gage d'universalité dans la mesure où il n'est pas influencé par une méthodologie qui risquerait à terme de l'orienter vers des domaines spécifiques comme l'informatique. Pour une approche orientée objet il faut donc se tourner vers des méthodes selon le contexte d'utilisation, voire les mixer pour obtenir une modélisation cohérente.

2.3.1.1 Modélisation de la structure du système

Dans cette partie du standard sont décrits les concepts permettant de caractériser la structure d'un système ainsi que les méta-modèles régissant le cadre d'utilisation de ces concepts au sein des diagrammes. Les concepts sont regroupés en *paquetages*, plus communément utilisés sous le terme anglophone *packages*, et sont décrits dans des ***diagrammes de structure*** dont les variations permettent de définir des diagrammes plus spécialisés en fonction des éléments que l'on souhaite représenter (classes, composants, structures composites, packages, objets).

Les classes. Entité la plus utilisée dans les processus de modélisation, elle permet de stocker les caractéristiques d'un objet donné en définissant ses attributs et ses méthodes, mais également de préciser les interactions avec l'environnement extérieur avec la

spécification d'interfaces d'échange, déclarant des services que l'objet fournit ou requiert lors de son fonctionnement. Le ***diagramme de classes*** reçoit les représentations de ces classes et permet également de lier les classes entre elles avec des relations conformes au paradigme objet: la généralisation, la composition, un exemple est donné en Figure 2.11.

Une fois les classes reliées entre elles à un niveau structurel, il est possible de les préciser individuellement par l'intermédiaire d'interfaces d'entrée/sortie. Ces interfaces sont des concepts UML qui héritent désormais des classes UML. Elles possèdent donc des propriétés et des méthodes. On les utilise pour préciser les services requis par une classe pour qu'elle puisse fonctionner correctement, ainsi que les services fournis qu'elle met à disposition du reste du système.

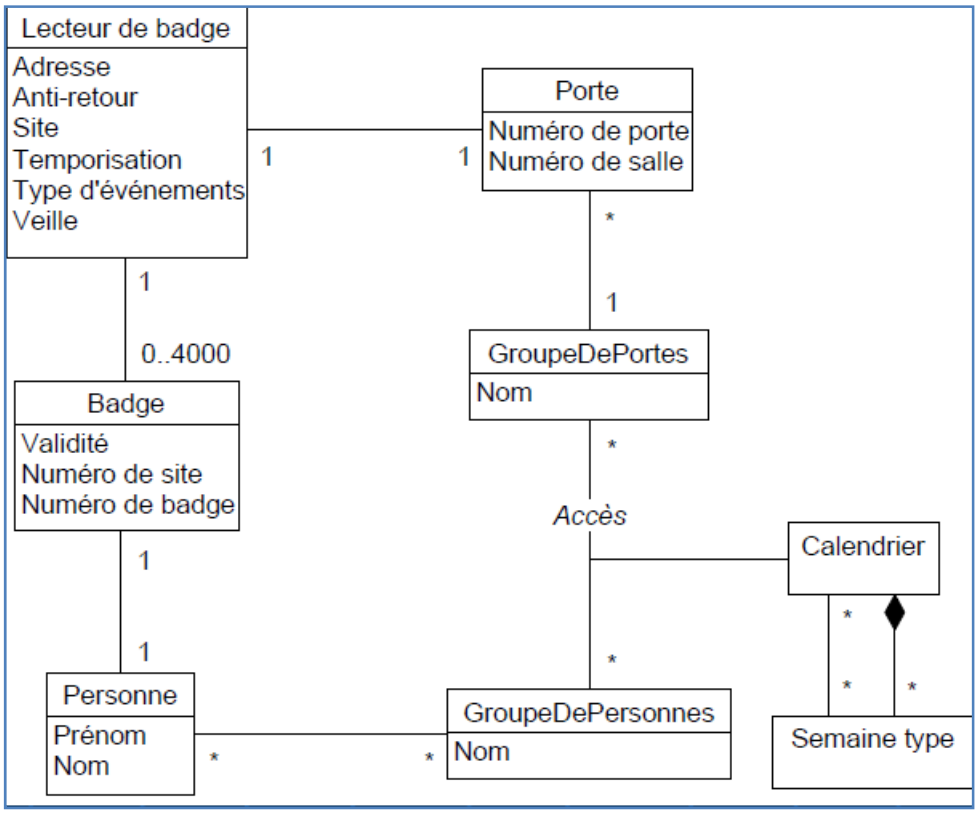


Figure 2.11 Diagramme de classes (contrôle des accès d'un bâtiment)

Les composants. Ils héritent directement des classes et représentent des structures de taille variable dont la principale différence par rapport aux classes est qu'ils peuvent être considérés comme des éléments autonomes et ne sont accessibles que via les services de leurs interfaces. On peut les rapprocher au niveau fonctionnel des composants COM, Java Beans, CORBA ou .NET.

Les structures composites. Elles font partie des nouveautés importantes apportées par la version 2.0 du langage **UML**. Elles autorisent une description beaucoup plus détaillée et cohérente de la façon dont les objets et les instances de composants peuvent être encapsulés dans la spécification d'éléments parents mais également des relations entre les services ou méthodes fournis à un niveau d'abstraction supérieur et ceux fournis par les éléments internes

Les artefacts et les nœuds de déploiement. Un ***artefact*** est un élément concret d'information produit par le processus de modélisation ou utilisé par le système décrit. Par exemple, les fichiers sources de code informatique qui peuvent être générés dans un langage donné sont des artefacts potentiels, de même qu'un message mail ou un fichier de stockage XML.

Les ***nœuds de déploiement*** UML représentent des ressources de stockage ou d'exécution ayant la possibilité d'accueillir des artefacts ou bien des instances de composant.

On utilise un ***diagramme de déploiement*** pour spécifier les artefacts et les nœuds. Il permet de représenter notamment une architecture physique ou informatique d'accueil pour les composants et les artefacts spécifiés dans la modélisation. Les nœuds peuvent être connectés entre eux formant ainsi des réseaux qu'il est possible de spécifier pour décrire précisément l'architecture du système étudié.

Les collaborations. Parfois, un concepteur peut vouloir représenter une relation au niveau de la structure sans connaître les objets spécifiques composant la relation. Pour permettre la modélisation de cette relation, UML propose le concept de collaboration. A défaut de pouvoir connecter des objets précis avec une relation, on définit des interfaces pour expliciter des « *rôles* » qui devront être joués par les éléments que l'on voudra connecter par la suite.

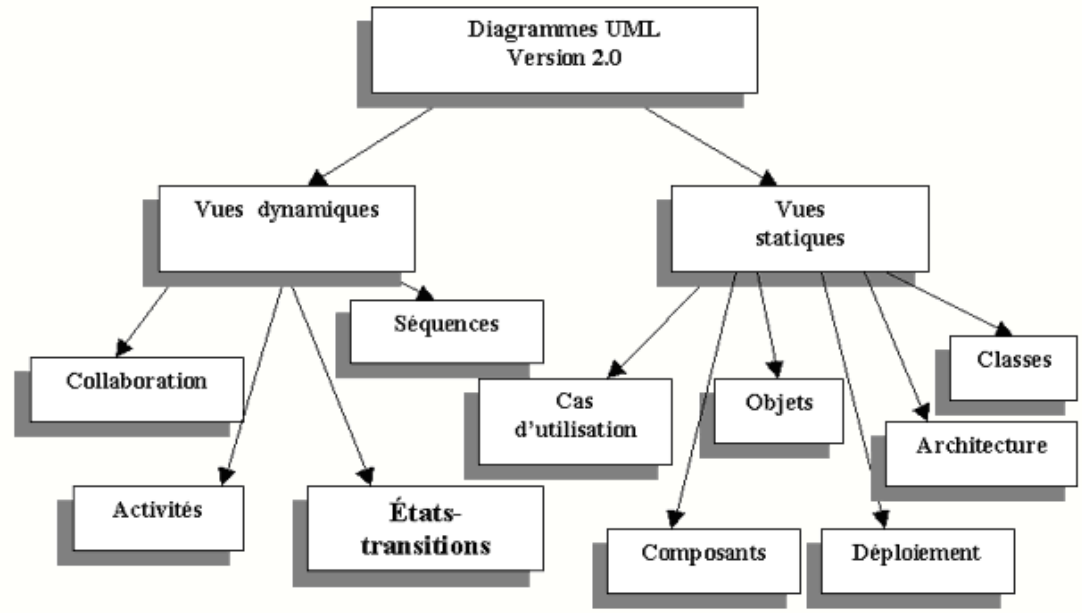


Figure 2.12 Les différents diagrammes d'UML

2.3.1.2 Modélisation du comportement du système

Avant d'introduire des diagrammes, **UML** spécifie en premier lieu des concepts permettant d'exprimer des caractéristiques dynamiques. On fait la distinction entre les actions, les activités, les états, les interactions et les cas d'utilisation. Nous allons passer brièvement en revue leur utilisation pendant le processus de modélisation et indiquer les liaisons effectuées avec le modèle structurel pour maintenir la cohérence globale du modèle.

L'objectif ici est de pouvoir détailler le comportement de tous les éléments décrits de façon statique dans la structure du système et de montrer comment ils interagissent pour réaliser les fonctions attendues de l'ensemble.

Les actions. Comme précisé dans le standard, une action représente l'unité élémentaire des spécifications comportementales d'un système. Elle possède un ensemble d'entrées et active ou envoie un certain nombre de sorties en fonction du traitement qu'elle effectue.

Tout le travail de description du comportement des éléments structurels va donc consister à organiser ces actions au sein de diagrammes de comportement de façon à produire le comportement dynamique souhaité pour l'élément étudié.

Les activités. Un peu à l'image des structures composites, une activité contient un ensemble d'actions reliées entre elles qui permettent de réaliser le comportement global de l'activité, sa réaction aux sollicitations extérieures, l'envoi ou le stockage d'informations. Les activités sont décrites dans des ***diagrammes d'activité***. Lorsqu'une; activité est exécutée, les actions internes de l'activité vont être exécutées une ou plusieurs fois selon la façon dont elles sont connectées entre elles et avec les entrées et les sorties de l'activité parente.

Les partitions. Elles permettent d'intégrer un niveau de description supérieur en regroupant les actions ou activités qui ont des caractéristiques communes par exemple si elles sont réalisées dans un même contexte particulier. Les partitions sont très pratiques, par exemple pour modéliser un flux d'information entre différents services ou sites géographiques d'une société. On partitionne alors selon les services et on intègre les actions de chacun en définissant les flux de contrôle et les flux d'objets correspondant aux échanges entre les différents services (par exemple envoi de mails dans le cas d'un flux d'objets, ou ordre simple dans un flux de contrôle).

Les interactions. Alors que les activités et les actions vont souvent être utilisées pour décrire le comportement interne d'un élément structurel, les interactions ont pour vocation de permettre à l'utilisateur de représenter les échanges et les sollicitations qui vont être effectués entre ces éléments structurels. De plus, les diagrammes proposés pour spécifier ces interactions vont permettre d'introduire des contraintes de séquencement entre ces interactions pour que la description du comportement des échanges entre les objets soit précisément modélisée.

Il existe cinq diagrammes permettant la modélisation des interactions d'un système, le diagramme de séquence (sequence diagram), le diagramme de vue générale des interactions (interactions overview diagram), le diagramme de communication (communication diagram), le diagramme temporel (timing diagram) et le diagramme de tables d'interaction (interaction tables). Il n'est pas obligatoire de tous les utiliser dans une même spécification; le choix sera à effectuer en fonction des attentes de l'utilisateur au niveau de la forme de représentation et du type de comportement dynamique qu'il veut décrire [Mul 00].

Les machines à états. Le diagramme de séquence s'appuie sur une description du comportement basée sur le séquencement des échanges de messages entre objets et le diagramme d'activité centre sa représentation sur un enchaînement d'actions ou d'activités. Les machines à états proposent une approche supplémentaire et complémentaire pour la modélisation en permettant de décrire le comportement *discret* des éléments structurels avec la modélisation des différents états d'un élément ainsi que les conditions et les actions associées au passage d'un état à un autre.

Les cas d'utilisation. Les cas d'utilisation UML permettent de définir, dans un diagramme de cas d'utilisation, les interactions sous une forme fonctionnelle entre les éléments extérieurs du système et ce dernier.

2.3.2 Rational Unified Process (RUP)

2.3.2.1 Historique

Le processus unifié est le résultat de la fusion des méthodes Objectory d'Ivar Jacobson, Booch de Grady Booch et OMT de James Rumbaugh, enrichi de nombreux apports issus des travaux d'élaboration du standard UML. La figure 2.13 retrace la succession des produits qui en sont issus, depuis le processus Objectory, dont la première version est sortie en 1987, jusqu'au Rational Unified Process (sortie en 1998) en passant par le Rational Objectory Process (1997). Le Rational Unified Process a évolué au fil des années pour finir par incarner l'expérience de milliers de projets dans tous les domaines imaginables. Historiquement, le RUP est le successeur direct de Rational Objectory Process (version 4) (Figure 2.13).

2.3.2.2 Le RUP et le Développement Itératif

La plupart des équipes de développement logiciel continuent à appliquer un processus en cascade, dans lequel chaque phase observe une séquence stricte : spécification, analyse et conception, implémentation et intégration, et enfin les tests. On rencontre couramment une méthode en cascade modifiée, qui ajoute des boucles de rétroaction à ce flux global.

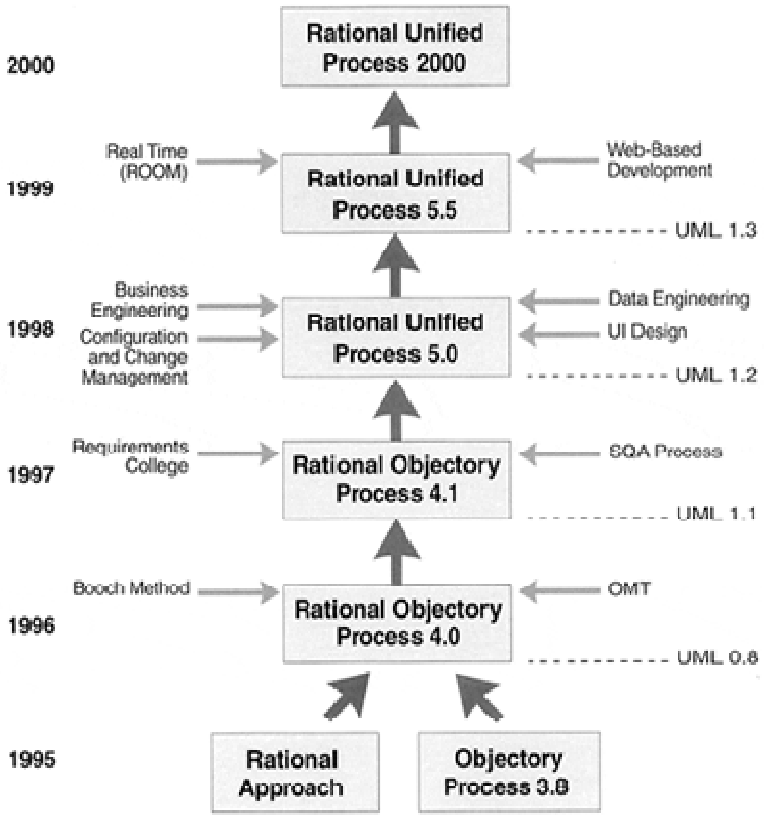


Figure 2.13 Historique du RUP [Kru 00]

De telles approches forcent des membres clés de l'équipe à de longues périodes d'inactivité, et diffèrent les tests en fin de cycle de vie du projet, moment où les problèmes ont tendance à être plus difficiles et plus coûteux à résoudre. Ils constituent ainsi de sérieuses menaces par rapport aux délais de livraison. A l'inverse, le RUP applique une démarche itérative, autrement dit une suite d'étapes incrémentales, ou itérations. Chaque itération comprend la plupart des disciplines du développement, sinon toutes (spécification, analyse, conception, implémentation, etc.), comme le montre la figure 2.14. Elle a un ensemble d'objectifs bien défini et produit une implémentation fonctionnelle partielle du système final. Chaque itération successive s'appuyant sur le travail effectué dans les précédentes, le produit final évolue et se raffine jusqu'à ce qu'il soit parachevé.

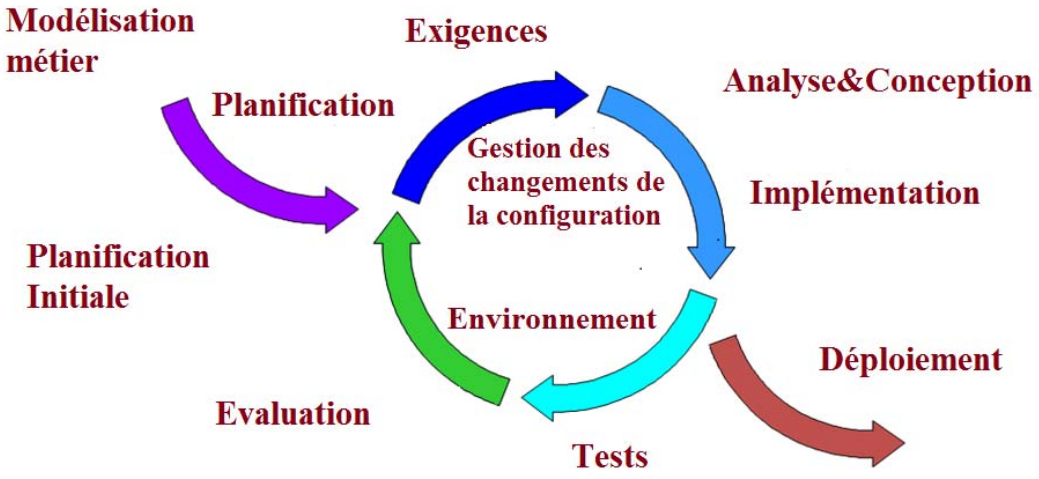


Figure 2.14 RUP et le développement itératif

L'approche itérative s'est montrée supérieure à la méthode en cascade pour plusieurs raisons :

- Elle s'adapte à l'évolution des besoins.
- L'intégration ne se résume pas à un « big bang » en fin de projet.
- Les risques sont généralement décelés lors des premières intégrations.
- Le management dispose d'un moyen d'apporter des modifications tactiques au produit.
- La réutilisation est facilitée.
- Les anomalies sont détectées au cours de plusieurs itérations.
- Les compétences sont mieux exploitées.
- L'apprentissage est permanent.
- Le processus de développement lui-même s'améliore continuellement.

2.3.2.3 Architecture du Rational Unified Process

RUP a été conçu grâce à des techniques analogues à celles de la conception logicielles. Il a été en particulier modélisé en appliquant le modèle SPEM (Software Process Engineering Metamodel) — un standard de modélisation de processus basé sur UML (Unified Modeling Language). La figure 2.15 représente l'architecture globale de RUP. Le processus possède deux structures ou (deux dimensions) :

- ***Une structure dynamique.*** La dimension horizontale représente la structure dynamique, autrement dit la dimension temporelle du processus. Montre comment

celui-ci, exprimé en termes de cycles, de phases, d'itérations et jalons, se déroule au cours de la durée de vie du projet.

- ***Une structure statique.*** La dimension verticale représente la structure statique du processus. Elle décrit la façon dont ses éléments – activités, disciplines, artefacts et rôles – sont regroupés logiquement en disciplines principales, ou enchaînements d'activités (workflow).

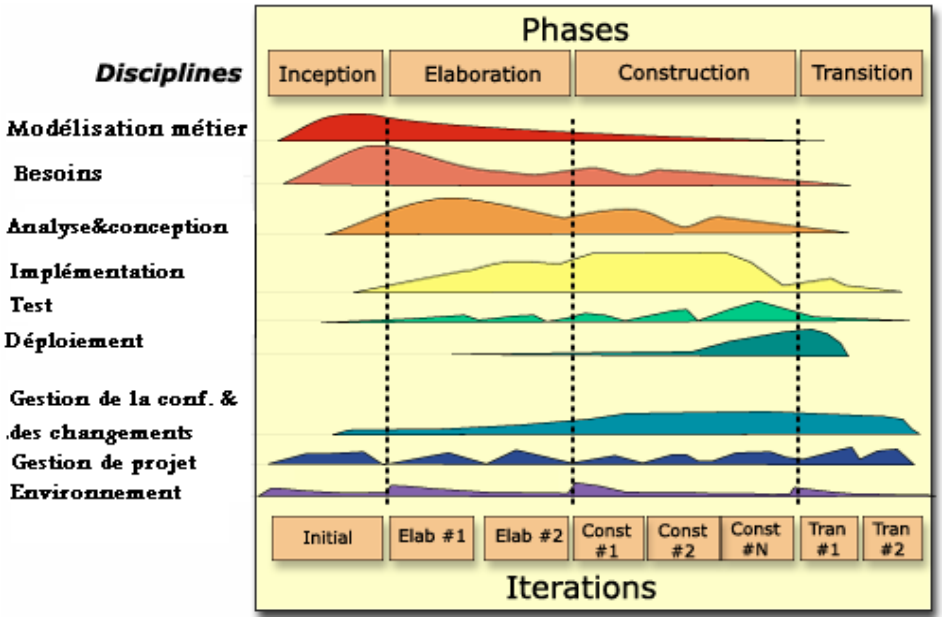


Figure 2.15 Les deux Dimensions de RUP

A - La structure dynamique du RUP(Cycle de vie du RUP: phases, objectifs et jalons)

A.1 - Concepts fondamentaux

Cycles. Un cycle de développement est le laps de temps qui s'écoule entre le tout début du projet et la livraison du produit (ou l'annulation du projet). Il comprend toutes les activités exécutées durant cette période.

Phases. Chaque cycle du RUP est décomposé en une suite de quatre **phases**, nommées Inception, Elaboration Construction et Transition.

Itérations. Chaque phase peut comprendre une ou plusieurs itérations. Chaque itération développe une partie du logiciel -- une version interne ou externe – et se conclut par un jalon mineur, qui constitue un point permettant d'évaluer l'avancement du projet. Le produit logiciel croît de façon incrémentale au fur et à mesure des itérations.

La structure dynamique concerne le cycle de vie, ou dimension temporelle, d'un projet. Le RUP applique une approche structurée au développement itératif, divisant un projet en quatre phases : inception, élaboration, construction et transition.

A.2 - Jalons des phases du cycle de vie du RUP.

Le cycle de vie du RUP compte quatre phases: inception, élaboration, construction et transition. Il se termine avec la livraison d'un produit logiciel complet. Chacune des quatre phases du RUP possède un jalon et un ensemble bien défini d'objectifs. Ceux-ci nous permettent de déterminer les activités à effectuer et les artefacts à produire (Figure 2.16).

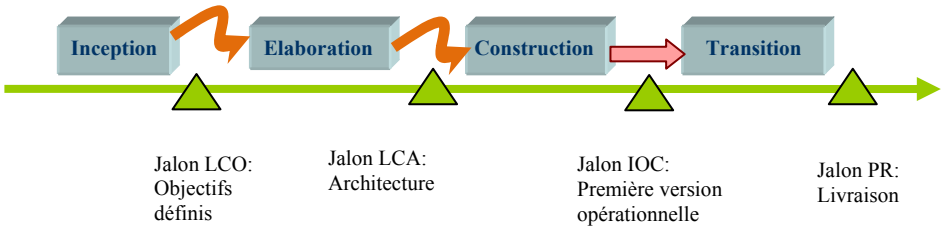


Figure 2.16 Les jalons des phases du cycle de vie du RUP

Chaque itération doit être précisément centrée sur ce qui est strictement nécessaire pour atteindre les objectifs métiers de la phase en question. Chaque phase comprend une ou plusieurs itérations, consacrées à la production des livrables techniques requis pour atteindre les objectifs métiers de cette phase.

A.3 - Les phases du cycle de vie du RUP.

Le cycle de vie du RUP compte quatre phases : inception, élaboration, construction et transition. Il se termine par une livraison d'un produit logiciel complet. Même si ces quatre phases sont exécutées séquentiellement, son cycle de vie est fondamentalement **itératif et piloté par les risques**. Il est vrai que les premières semaines ou les premiers mois d'un projet mettent plutôt l'accent sur les exigences et qu'il y aura plus de tests et de finitions en fin de projet. C'est précisément ce changement de priorité au cours du cycle de vie que représentent les « bosses » sur le graphique des itération (voir Figure 2.15), et chacune d'elles comprend de nombreuses activités de développement produisant du code testé qui devient une version interne, et plus tard externe.

Phase Inception.

L'inception est la première des quatre phases du cycle vie du RUP. Elle est entièrement dédiée à l'étude du périmètre et des objectifs du projet, et doit permettre de récolter suffisamment d'information. Elle vise les objectifs suivants.

Objectif 1 : Comprendre le système à construire. Déterminer la vision, le périmètre du système et ses limites, autrement dit ce qui en fait partie et ce qui n'en fait pas partie. Identifier les personnes qui veulent le système et la valeur qu'elles lui accordent.

Tous les intervenants (clients, responsables, analystes, développeurs, rédacteurs techniques et autres personnes impliqués) doivent s'accorder sur une définition commune du succès. Pour ce faire, voici les mesures à prendre :

- **Convenir d'une vision de haut niveau**. La solution s'exprime par la production d'un document. Pour un petit projet, ce document peut être informel, voire se réduit à un message qui résume une précédente discussion autour d'un tableau blanc. Pour les projets de taille moyenne, ce document peut comporter quelques pages.
- **Fournir une vision large et superficielle du système**. Décrire ce que le système doit faire sans trop détailler, pour éviter de s'enliser ou de voir certains intervenants, responsables ou clients par exemple, perdre de vue l'essentiel de l'information parce qu'elle est enfouie sous une masse de documentation.
- **Détailler quels sont les acteurs et les cas d'utilisation essentiels** pour que tous les intervenants les comprennent facilement et que les membres de l'équipe puissent les utiliser directement.

Objectif 2 : Identifier la fonctionnalité essentielle du système. Il importe de décider quels sont les cas d'utilisation essentiels ou les plus significatifs du point de vue architectural pour pouvoir consacrer plus de temps dès le départ aux points les plus critiques. Décider quels sont les cas d'utilisation (les interactions avec les systèmes) les plus critiques.

Chef de projet et architecte doivent collaborer étroitement, impliquer tous les autres intervenants nécessaires et recourir à plusieurs critères pour déterminer quels sont les cas d'utilisation les plus critiques.

La fonctionnalité est le noyau de l'application, ou elle représente des interfaces clés, et a donc un impact majeur sur l'architecture. Généralement un analyste identifie ces cas d'utilisation en analysant la stratégie de gestion de redondance, les risques de contention des ressources, les risques de performances, les stratégies de sécurité des données, etc.

La fonctionnalité doit être livrée. La fonctionnalité résume l'essentiel du système, et livrer une application dont elle serait absente n'aurait pas de sens. En général, les experts du domaine savent de quelle fonctionnalité il s'agit du point de vue de l'utilisateur (comportement primaires, transaction de données en pics, transactions de contrôle critiques, etc.).

La fonctionnalité traite un élément de l'architecture qui n'est couvert par aucun cas d'utilisation. Il faut s'assurer de traiter tous les risques majeurs et comprendre chaque élément du système. Même si un élément donné de l'architecture semble présenter peu de

risques, il peut cacher des difficultés techniques inattendues, qu'il est possible de détecter en concevant, implémentant et testant une partie de la fonctionnalité qui lui est associée.

Les critères 1 et 3 sont particulièrement importants pour l'architecte, tandis que les chefs de projet se concentreront particulièrement sur les points 1 et 2.

Dans un système comprenant vingt cas d'utilisation, généralement trois ou quatre d'entre eux sont critiques. Dans certains systèmes, un ou deux cas d'utilisation peuvent constituer le cœur de l'application, et les autres sont des cas d'utilisation "auxiliaires" permettant leur exécution. Dans ce cas, moins de 20 à 30% des cas d'utilisation ont une signification architecturale, et l'équipe n'implémentera généralement que quelques scénarios pour ceux qui font partie du noyau.

Objectif 3 : Déterminer au moins une solution possible. L'objectif global de la phase d'inception est de déterminer si la poursuite du projet est possible, et de s'assurer de l'existence d'au moins une architecture potentielle permettant de construire le système sans trop de risques et avec un coût raisonnable. Dans certains cas, il faut acquérir ou implémenter certains éléments clés de l'architecture, ou d'autres suggestions d'architecture, afin de mieux comprendre les risques auxquels l'équipe est confrontée et les options dont elle dispose. Dans le cas d'application où les différents intervenants peuvent avoir du mal à se représenter le système final, l'équipe doit également consacrer un certain temps à développer quelques prototypes fonctionnels, suffisamment riches pour vérifier que la vision a un sens. A la fin de la phase d'inception, l'équipe doit avoir une idée globale des risques auxquels elle est confrontée.

Objectif 4 : Comprendre les coûts, les délais et les risques associés au projet. Comprendre le système à construire est essentiel, mais définir la façon de le construire et à quel coût est également crucial. Pour déterminer s'il faut poursuivre un projet, il faut avoir une idée globale de son coût. La plupart des coûts sont associés aux ressources dont l'équipe aura besoin et au temps nécessaire pour terminer le projet. En combinant ces données avec ce que l'équipe connait des fonctionnalités nécessaires et ce qu'elles représentent pour les utilisateurs, elle est à même de construire l'étude de rentabilité. Cette dernière documente la valeur économique du projet exprimée en termes quantitatifs, de retour sur investissement (ROI) par exemple. C'est l'instrument qui permet d'obtenir un financement adéquat. Il décrit également les principaux risques non réduits, et donc le degré d'incertitude qui demeure.

Objectif 5 : Décider du processus à appliquer et des outils à utiliser. Il importe que l'équipe partage une vision commune de la façon dont elle va développer le logiciel, autrement dit du processus qu'elle va appliquer. Il faut s'assurer que le processus est aussi léger que possible pour minimiser les surcoûts, et qu'il répond aux besoins spécifiques du projet. Un petit projet supporte que les décisions sur le processus exact à suivre soient prises à

mesure, mais un projet plus important nécessite plus de temps préalable à consacrer à son choix.

L'idée est d'obtenir lors de la première itération un processus et un environnement d'outils fonctionnels. En deuxième itération, le processus et les outils seront déployés, et on obtient un feed-back immédiat sur ce qui fonctionne et ce qui ne fonctionne pas. En fonction de ce *feed-back*, on actualise le processus et ses outils, et on continue à itérer jusqu'à ce qu'on soit satisfait de son environnement.

Une fois décidé d'un processus, on peut sélectionner des outils. Dans certains cas, l'environnement sera prédéterminé, pour correspondre par exemple à un standard de l'entreprise. Sinon, on doit choisir notamment un environnement de développement intégré (EDI), un outil de modélisation graphique, un outil de gestion de la configuration et des changements. Il importe que ces outils permettent d'automatiser efficacement le processus qu'on a choisit. On devra peut-être alors personnaliser les outils et les modèles, installer différents répertoires pour le projet, etc.

Revue de projet : le jalon LCO (objectifs définis). La phase inception se termine par le premier grand jalon du projet, le jalon LCO. A ce stade, on examine les objectifs liés au cycle de vie du projet. Si ce jalon n'est pas atteint, le projet doit être annulé ou reconsidéré. S'il est voué à l'échec, mieux vaut en rendre compte tôt que tard, et l'approche itérative combinée à ce jalon peut révéler que c'est son destin.

Le projet peut être annulé ou considérablement repensé s'il échoue à passer ce cap.

Phase Elaboration

L'élaboration est la deuxième des quatre phases de l'approche RUP. Son objectif est de définir l'architecture du système et de l'établir comme référentiel afin de fournir une base stable au gros de l'effort de conception et d'implémentation en phase de construction. Celle-ci émerge de l'analyse des exigences les plus significatives et de l'évaluation des risques.

Cet objectif général se traduit par quatre sous-objectifs majeurs qui traitent chacun une zone de risque (exigences, architecture, coût, calendrier, Etc.). Cela permet de passer à la phase de construction avec un minimum de risques et de problèmes.

Les objectifs de la phase élaboration sont :

- Objectif 1 : Comprendre en détail les exigences
- Objectif 2 : Concevoir, implémenter, valider l'architecture et en établir une version de référence.
- Objectif 3 : Réduire les risques essentiels et estimer plus exactement les délais et le budget.
- Objectif 4 : Affiner le plan de développement et mettre en place l'environnement de développement

Revue de projet : le jalon LCA. Le jalon LCA termine la phase d'élaboration. Parvenu à ce stade, on examine les objectifs détaillés du système et son périmètre, le choix de l'architecture et la résolution des risques majeurs. Si le projet n'atteint pas ce jalon, il peut être annulé, ou doit être au moins sérieusement reconsidéré. Combinée avec ce jalon, la démarche itérative force une telle décision.

Phase Construction

La construction est la troisième des phases du cycle de vie du RUP. Cette phase est centrée sur la conception détaillée, l'implémentation et les tests qui permettront d'étoffer le système et d'en faire un produit fini.

La construction est entièrement dédiée au développement dans des conditions rentables d'un produit complet – une version opérationnelle du système -- qui puisse être déployé dans la communauté des utilisateurs. Cela se traduit par les objectifs suivants :

- Objectif 1 : Minimiser les coûts de développement et obtenir un certain degré de parallélisme
- Objectif 2 : Développer d'une manière itérative un produit prêt à la transition vers la communauté des utilisateurs.

Revue de projet : le jalon IOC (Initial Operational Capability Milestone) : Architecture définie. La phase de construction se termine par un jalon important : le jalon IOC qui sert à déterminer s'il est possible de déployer le produit dans un environnement de test bêta, en répondant notamment aux questions suivantes :

- Cette version du produit est-elle stable et suffisamment mature pour être déployée dans la communauté des utilisateurs ?
- Tous les intervenants sont-ils prêts pour la transition ?
- Les dépenses en ressources comparées aux dépenses prévisionnelles sont-elle toujours acceptables ?

Si le projet n'atteint pas ce jalon, la phase de transition peut être différée en ajoutant une itération à la phase de construction.

Phase Transition

L'objectif de la phase **transition** est d'assurer que le logiciel réponde parfaitement aux besoins de ses utilisateurs. Elle se déroule normalement en une ou deux itérations, qui consistent à tester le produit en préparation de sa livraison, et à apporter quelques ajustements en fonction du *feed-back* des utilisateurs. A ce stade du cycle de vie, tous les problèmes structurels majeurs doivent être résolus, et ce feed-back ne doit pas porter principalement que sur des questions d'optimisation, de configuration, d'installation et d'ergonomie. La phase de transition des projets plus complexes peut comprendre plusieurs itérations, chacune produisant une version déployable plus élaboré du système.

- Objectif 1 : Comprendre en détail les exigences

- Objectif 2 : Former les utilisateurs
- Objectif 3 : Préparer le site de déploiement et convertir les bases de données opérationnelles
- Objectif 4 : Préparer le lancement
- Objectif 5 : Obtenir le consensus des intervenants
- Objectif 6 : Améliorer les performances futures

Revue de projet : le jalon PR: La phase de transition se termine sur le quatrième grand jalon, le jalon PR. Il permet de déterminer si les objectifs sont atteints et si on doit commencer un autre cycle de développement. A ce jalon, le produit est en production, et le processus d'exploitation, de maintenance et de support commence. Cela peut impliquer d'entamer un nouveau cycle de développement pour apporter des améliorations majeures, ou aller vers une version de maintenance pour corriger certains défauts.

B - La structure statique du RUP

La structure statique concerne la façon dont les différents éléments du processus –rôle, artefacts, activités et disciplines – sont logiquement regroupés en enchaînements d'activités (workflows). Un processus décrit ***qui*** fait ***quoi***, ***comment*** et ***quand***. Le RUP est représenté à l'aide de quatre éléments clés :

- Activités. Comment ?
- Artefacts. Quoi ?
- Enchaînements d'activités. Quant
- Rôles. Qui ?

Rôle. Un rôle définit simplement la façon dont un travail doit être effectué, et spécifie la compétence que les individus qui le jouent doivent posséder et les responsabilités qui leur incombent. Une personne joue généralement un ou plusieurs rôles, et plusieurs personnes peuvent jouer le même rôle.

Activités. Une activité liée à un rôle spécifique constitue une unité de travail qu'un individu, à qui ce rôle est attribué, peut être amené à exécuter. Elle a une fin explicite, habituellement exprimée en termes de création ou d'actualisation d'un artefact donné, par exemple un modèle, un composant ou un plan. Chaque activité est affectée à un rôle spécifique. Une activité doit pouvoir servir d'éléments de planification et de mesure de la progression.

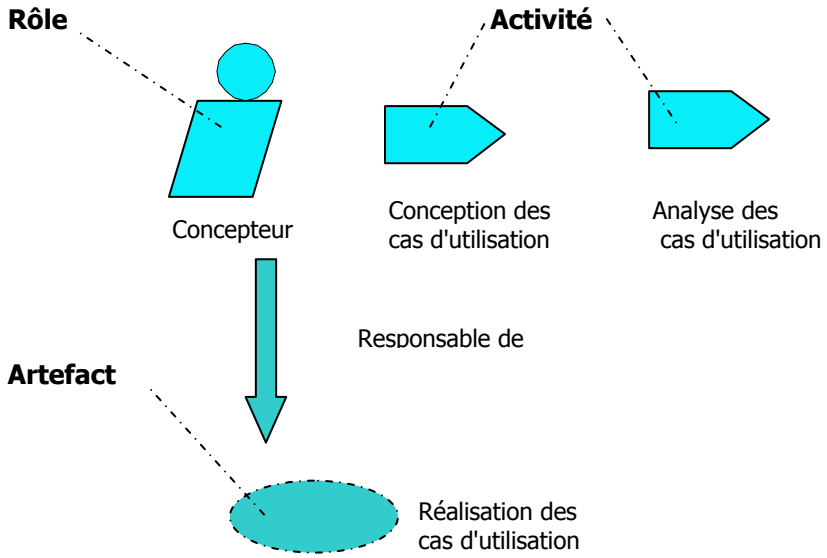


Figure 2.17 Rôles, activités et artefacts

Etapes. Les activités sont décomposées en étapes qui se divisent en trois grandes catégories :

- Les étapes de **réflexion**, où la personne qui joue le rôle étudie la nature de la tâche, recueille et examine les artefacts entrants, et formule un résultat à atteindre.
- Les étapes de **réalisation**, où le rôle crée ou met à jour des artefacts.
- Les étapes d'**évaluation**, où le rôle inspecte le résultat en fonction de certains critères.

Artefacts. Les **artefacts** sont des unités d'information produites, modifiées ou utilisées par un processus. Ce sont les éléments tangibles du projet, ceux qu'il produit ou exploite pour parvenir au produit final. Les artefacts sont utilisés en entrée par des rôles pour réaliser une activité et sont le résultat (ou la sortie) d'autres activités. Ils prennent différentes formes :

- un ***modèle***, comme le modèle des cas d'utilisation ou le modèle de conception ;
- un ***élément de modèle***, comme la vision ou l'étude de rentabilité ;
- un programme source ;
- un ***exécutable***, par exemple un prototype exécutable.

Un artefact peut être documenté de façon formelle (à l'aide d'un outil spécial ou informelle (dans un message électronique ou un tableau blanc).

Enchaînements d'activités. La simple liste de tous les rôles, activités et artefacts, ne constitue pas encore un processus. Il faut disposer de moyen de décrire des séquences significatives d'activités qui produisent un résultat, et de mettre en évidence les interactions entre rôles. C'est précisément l'objectif des enchainements d'activités (Workflows). Ils se présentent sous plusieurs formes, les deux plus courantes étant des ***disciplines***, qui sont des enchaînements d'activités de haut niveau, et les ***détails d'enchainement d'activité*** qui sont des enchaînements d'activités au sein d'une discipline. UML permet d'exprimer un enchaînement d'activités sous forme de diagramme de séquences, de collaboration ou d'activité [Kro 05].

Autre éléments du processus. Les rôles, les activités (organisées en enchaînements d'activités) et les artefacts représentent l'épine dorsale de la statique du RUP. Mais certains autres éléments s'y ajoutent, pour rendre le processus plus facile à comprendre et mettre en œuvre et mieux guider les praticiens (Figure 2.18). Les éléments supplémentaires sont les suivants :

- des **principes et conseils**, qui fournissent des règles, des recommandations ou des heuristiques pour appuyer la réalisation des activités, des étapes et des artefacts ;
- des **modèles** (Templates) pour les différents artefacts ;
- des **guides d'utilisation d'outils**, pour établir un lien avec les outils de développement et fournir une aide à leur utilisation ;
- des **concepts**, pour présenter les définitions et les principes clés ;
- des **cartes**, pour guider l'utilisateur d'un point de vue donné.

Disciplines. Tous les éléments du processus—rôles, activités, artefacts, concepts associés, lignes directrices et modèles—sont regroupés dans des conteneurs logiques nommés **disciplines**. Le produit RUP standard compte neuf disciplines (voir l'encadré suivant). La liste n'est pas définitive, et toute entreprise peut apporter des extensions au RUP en ajoutant des disciplines.

Voici les neufs disciplines :

1. Modélisation métier
2. Gestion des exigences
3. Analyse et conception
4. Implémentation
5. Déploiement
6. Tests
7. Gestion de projet
8. Gestion de la configuration et changements
9. Environnement

Bien que les noms des six disciplines noyau d'ingénierie puissent évoquer les phases séquentielles dans un processus traditionnel en cascade, nous devrions maintenir dans l'esprit que les phases d'un processus itératif sont différentes et que ces enchaînements d'activités sont revisités à plusieurs reprises tout au long du cycle de vie. L'enchaînement d'activités complet actuel d'un projet intercale (Interleaves) ces neuf enchaînements d'activités noyau, et les répète durant les différentes phases avec une intensité à chaque itération (Figure 2.19).

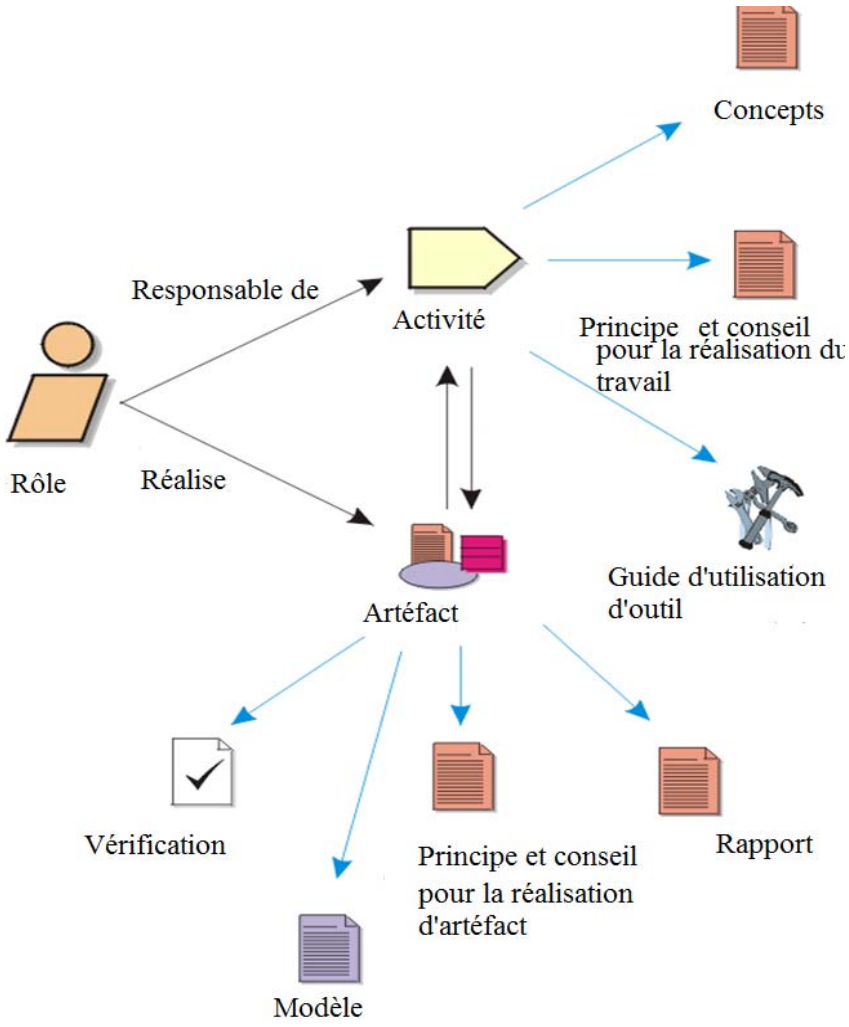


Figure 2.18 Aperçu sur les concepts du RUP [Kru 00]

2.3.2.4 Le RUP- Un produit personnalisable

Chaque projet et chaque entreprise ont des besoins uniques qui nécessitent un processus adapté à leur situation spécifique. Pour répondre à cette exigence, le produit RUP constitue un processus générique complet composé de plusieurs parties intégrées.

- **Meilleure pratique.** Le RUP est accompagné d'une bibliothèque de meilleures pratiques, produites par IBM Software et ses partenaires.

- **Outils de mise à disposition de processus.** Le RUP est littéralement à portée de main des développeurs, parce qu'il est livré en ligne par l'intermédiaire de la technologie Web et non sous forme de livres ou de dossiers.
- **Outils de configuration.** La forme modulaire et électronique du RUP offre la possibilité de le personnaliser et de le configurer pour répondre aux besoins spécifiques d'une organisation de développement.
- **Outils de création de processus RPW** (Rational Process Workbench)
- **Communauté/Marché.** La communauté virtuelle RDN *(Rational Developer Network)* permet aux utilisateurs d'échanger leurs expériences avec leurs pairs et avec des experts, et d'accéder aux dernières informations.

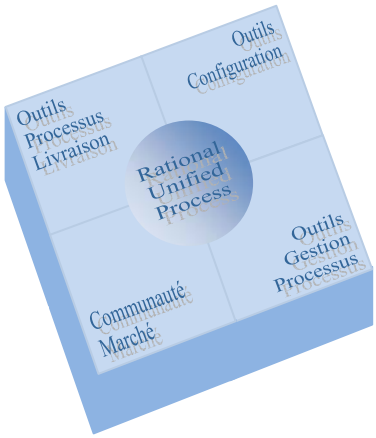


Figure 2.19 Le processus générique du RUP

2.4 Démarches de Développement Actuelles et Ontogénèse

L'évolution est un terme qui désigne tout changement appliqué à un modèle. A travers une littérature abondante, le terme évolution est utilisé comme synonyme de changement ou modification. Actuellement, il n'existe pas de définition standard de l'évolution. Alors que certains lui donne une portée large en considérant que l'évolution d'un système touche toute les phases : aussi bien le développement que la maintenance, d'autres l'utilise comme substitut préférable de la maintenance [Mes 06].

Par définition l'évolution non anticipée des logiciels n'est pas quelque chose qu'on puisse préparer lors de la conception d'un système logiciel. Par contre, l'évolution anticipée est prise en charge durant le développement du système pour être exécutée après sa mise en œuvre.

Pour rester utile, un système logiciel doit constamment évoluer. Ceci est principalement dû au changement et à l'accroissement des exigences de ses utilisateurs. Faire évoluer un

système est un réel challenge impliquant la satisfaction des besoins suivants : découvrir la partie du logiciel concernée par cette évolution, trouver un moyen de la réaliser sans entraîner la régression du système et enfin, pouvoir valider cette évolution. Un autre challenge serait de répondre aux besoins cités précédemment au moindre coût.

Modéliser, même partiellement, le processus de l'évolution par des concepts et des mécanismes qui lui sont dédiés semble nécessaire. Bien que plusieurs approches soient actuellement proposées comme des solutions aux problèmes de l'évolution, elles demeurent néanmoins des palliatifs. Très peu sont les approches qui redonnent à l'évolution un statut important et un paradigme de base, nous pouvons citer à titre d'exemple : "autonomic computing" [Mur 04], les systèmes biomorphiques [Lod 04], le projet « Reconfigurable POEtic Tissue » [Tem 02], et le paradigme Mage [Mes 04, Mes 06].

A notre connaissance, sur le plan des démarches de développement, il n'en existe pas encore qui soient dédiées à ces approches. On trouve dans la littérature plusieurs approches qui tentent de modéliser les évolutions. Néanmoins, toutes ses recherches [Eck 96, Eti 04, Sal 04, Pim 11], reste au niveau de l'analyse des besoins.

2.5 Conclusion

Au fil des années, plusieurs modèles de développement logiciel on été élaborés. Or, quand on pense au modèle en cascades, au modèle en spirale ou un modèle itératif, force est de constater que le développement logiciel est difficile à maîtriser. Des années 1970 à 2000, les processus de développement étaient principalement prédictifs et basés sur la production d'artefacts, souvent dans le but de satisfaire des normes. À l'opposé, principalement après les années 2000, les méthodologies agiles, qui sont réactives et qui mettent l'accent sur les ressources humaines ont gagné en popularité. Nous avons présenté dans ce chapitre RUP, une des méthodes agiles jouissant d'une notoriété mondiale et qui fait le noyau de la démarche proposée dans ce travail de thèse. Nous avons opté pour l'extension du RUP l'adjonction de deux autres disciplines. Le chapitre suivant constitue un état de l'art sur l'ingénierie des besoins des systèmes logiciels et une description de modèles existants dans la littérature pour l'anticipation des changements/évolutions des besoins.

CHAPITRE 3
INGENIERIE DES EXIGENCES

L'analyse des besoins est vue comme l'une des étapes cruciale dans la conception et le développement des systèmes logiciels car elle aborde le problème critique de concevoir le bon logiciel pour le client. Cette activité est plus difficile, voire complexe, pour le développement des systèmes ontogénétiques car l'analyse d'une situation courante est moins problématique que celle du futur. Néanmoins, il existe des approches prometteuses qui ont participé à réduire l'écart entre l'ingénierie des besoins des systèmes logiciels et la prise en compte des évolutions anticipées dans le cycle de développement.

Nous nous intéressons dans ce chapitre à la présentation de l'ingénierie des besoins et de quelques approches représentatives de cette phase.

3.1 L'ingénierie des Besoins : Une Etape Clé

Actuellement les systèmes logiciels sont devenus plus complexes, et leur développement est plus difficile et par conséquent leur évolution devient aussi fastidieuse. La mesure primaire du succès d'un système logiciel est le degré auquel il réalise le but pour lequel on l'a construit. Le processus d'Ingénierie des Besoins est un ensemble de tâches bien plus complexe que la simple production en début de projet d'un document spécifiant le système à construire. Définir les besoins pour un logiciel est un processus qui fait appel à des compétences humaines, techniques et méthodologiques très variées, alliant rigueur et créativité. Une étude efficace des besoins réduit très sensiblement le coût du développement et de la maintenance d'une application et accroît sa qualité [Aur 05].

Dans les méthodes d'ingénierie des exigences, deux types d'exigences peuvent être identifiés. Les exigences fonctionnelles qui spécifient les fonctions que le système à concevoir doit être en mesure d'effectuer. Les exigences non-fonctionnelles qui capturent les propriétés ou les contraintes sous lesquelles le système à concevoir doit fonctionner, telles que les aspects de performance, de qualité ou de sécurité. Les pratiques industrielles actuelles consistent à spécifier les exigences fonctionnelles dès les premières phases de développement logiciel et à laisser la prise en compte des exigences non-fonctionnelles au

niveau de l'implémentation. Ce choix est souvent justifié par l'énorme pression (en termes de temps) pour le déploiement rapide d'un prototype du logiciel [Cys 03].

3.1.1 Définitions

L'expression "Requirements Engineering" semble avoir été utilisée pour la première fois par [Hag 88]. Une traduction exacte devrait être "ingénierie des exigences" mais nous suivons l'usage en utilisant "ingénierie des besoins".

Tous les projets commencent par l'étape des besoins. Les besoins décrivent ce qu'un produit logiciel doit exécuter. Un besoin se rapporte typiquement à un certain aspect d'un nouveau produit ou un service ajouté [Aur 05].

Dans les sciences informatiques, le terme "ingénierie des besoins" est à la fois utilisé pour désigner un thème de recherche, pour référencer une activité du processus de développement de systèmes logiciels ainsi que pour référencer un "processus " de construction d'une spécification des besoins. Plusieurs définitions du terme "ingénierie des besoins", que nous énumérons ci-dessous, ont été introduites dans la littérature :

Le standard IEEE 610.12-1990 [ANS 90] définit un besoin comme étant :

« Une condition ou capacité nécessaire à un utilisateur pour résoudre un problème ou atteindre un objectif », ou comme étant « une condition ou capacité qui doit être reconnue ou possédée par un système pour satisfaire un contrat, un standard, une spécification ou d'autres types de documents formellement imposés ».

On trouve dans [Zav 93] une des définitions les plus précises de l'ingénierie des besoins :

«L'ingénierie des besoins est la branche du génie logiciel concernée par des buts du monde réel pour des fonctions des systèmes logiciels et des contraintes sur ces derniers. Elle est également concernée par la relation de ces facteurs pour préciser des spécifications du comportement du logiciel et leur évolution avec le temps et à travers des familles de logiciels».

Cette définition est attrayante pour certaines raisons. D'abord, elle accentue l'importance sur « les buts du monde réel » qui motivent le développement d'un système logiciel. Puisqu'elle représente aussi bien le "Pourquoi" que le "Quoi" d'un système (Figure 3.1). Par ailleurs, elle se rapporte « à des spécifications précises ». Celles-ci constituent la base d'analyse des besoins, validant ce que les parties prenantes veulent, définissant ce que les concepteurs doivent construire, et vérifiant qu'ils ont fait le travail correctement à la livraison.

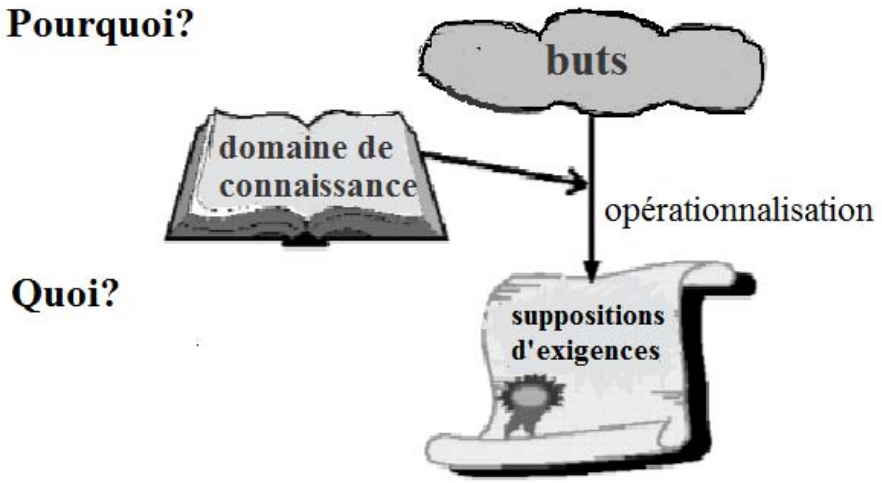


Figure 3.1 Le fondement de l'ingénierie des besoins [Rol 03]

3.1.2 Classification des Exigences

La littérature établit une distinction entre différents types de besoins, dans la pratique il n'est pas toujours facile d'identifier de telles différences [Ber 98]. Idéalement les besoins sont indépendants de la conception, montrant le "Quoi" que le système logiciel doit faire, plutôt que le "Comment" il doit le faire. Les besoins peuvent être classés dans plusieurs directions, comme illustré dans la table 3.1 [Aur 05].

Classification	Classe des besoins	Eclaircissement
1	Besoins fonctionnels	Ce que le système doit faire comme fonctions
	Besoins non-fonctionnels	Contraintes sur les types de solutions qui mènent les besoins fonctionnelles. Par exemple : précision, performance, sécurité et fiabilité
2	Besoins niveau but	Relié aux buts métier
	Besoins niveau domaine	Relié à l'espace du problème
	Besoins niveau produit	Relié au produit
	Besoins niveau conception	Quoi construire
3	Besoins primaires	Elicités à partir des parties prenantes
	Besoins dérivés des besoins primaires	Elicités à partir des besoins primaires
4	Besoins métiers	Besoins liés à l'activité de l'entreprise (son métier)
	Besoins processus	Comment les gents interagissent avec le système
	Besoins basés rôle	Par exemple: besoins consommateur, besoins utilisateurs, besoins IT, besoins système, et besoins de sécurité

Table 3.1 Classification des besoins

3.1.3 Activités de l'Ingénierie des Besoins

On trouve dans [Lam 09] et [Nus 00] une description des activités qui caractérisent une démarche d'ingénierie des exigences. Nous les présentons dans ce qui suit.

- **La compréhension du domaine**

Cette activité consiste en l'identification des parties prenantes à l'aide d'interviews en étudiant aussi l'environnement du système. Les problèmes du système actuel sont ainsi découverts et les possibilités d'amélioration sont analysées.

- **L'élicitation des exigences**

Cette activité consiste à découvrir les exigences et les hypothèses candidates pour le système en cours en se basant sur les faiblesses du système actuel, extraites dans l'activité de la compréhension du domaine. Plusieurs techniques peuvent être employées pour avoir les informations appropriées telles que les interviews avec les parties prenantes, les scénarios, les questionnaires, etc.

Nous avons sélectionné un noyau de huit techniques et approches qui offrent une couverture appropriée à travers la gamme de techniques et approches disponibles (par exemple **Ethnographie** comprend l'observation, et **JAD** est un exemple de travail actuel en groupe), et qui représentent à la fois l'état de l'art et l'état de la pratique.

La table suivante, tirée de [Aur 05], montre dans quelle phase les techniques d'élicitation peuvent être utilisées.

	Interview	Domaine	Groupwork	Ethnographie	Prototypage	Buts	Scenarios	Point de vue
Compréhension du domaine	X	X	X	X		X	X	X
Identification des sources des besoins	X	X	X			X	X	X
Analyse des parties prenantes	X	X	X	X	X	X	X	X
Sélection des techniques et approches	X	X	X					
Extraction des besoins	X	X	X	X	X	X	X	X

Table 3.2 Techniques et approches pour les activités d'extraction [Aur 05]

- **L'évaluation des exigences**

Le but de cette activité est d'évaluer les exigences afin de choisir les meilleures alternatives, en se basant par exemple sur l'analyse des risques et des conflits entre les exigences.

- **La spécification et la documentation des exigences**

Cette activité consiste à détailler, structurer et documenter les caractéristiques acceptées du système futur telles qu'elles ressortent de l'activité d'évaluation. Le produit obtenu est une première version du cahier des charges qui est écrit en langage naturel, semi-formel (des diagrammes) ou formel.

- **L'assurance de la qualité des exigences**

Le cahier des charges obtenu à l'issue de l'activité précédente doit être soigneusement analysé et validé avec les parties prenantes afin de repérer les insuffisances (incohérences, omissions, contradictions ou ambiguïtés) par rapport aux besoins réels. Diverses techniques peuvent être utilisées telles que les reviews, l'animation, etc. Le résultat final de cette activité est un cahier des charges consolidées.

- **La gestion de l'évolution des exigences**

Elle consiste à se préparer aux changements que le système futur peut subir. Cette anticipation d'évolution doit prendre en compte tous les aspects (les exigences, les hypothèses sur l'environnement, etc.) en utilisant diverses techniques telles que la gestion de la traçabilité, les changements à la volée, etc.

3.1.4 Importance de l'Ingénierie des Besoins

Le processus d'ingénierie des besoins tient une place importante à l'intérieur du processus de développement des systèmes logiciels. Le développement des spécifications des besoins de logiciel est largement identifié comme la base de la fonctionnalité du système. Les besoins d'un logiciel sont les causes déterminantes critiques de la qualité de ce logiciel. Des études empiriques prouvent que les erreurs dans les besoins sont les plus nombreuses pendant le cycle de vie de logiciel, elles sont aussi les plus chères et les plus longues à corriger. Le processus d'ingénierie des besoins tient une place importante à l'intérieur du processus de développement de systèmes logiciels.

Par exemple la plupart des problèmes de développement de système d'information se produisent lorsque le processus d'ingénierie de besoins n'est pas respecté (Figure 3.2). Fransiskus Adikara et al. ont catégorisé de nombreux risques des utilisateurs liés au processus d'exigences, à savoir: le risque de la connaissance, le risque de volatilité des exigences et les risques de la documentation [Adi 13].

- Le risque de connaissance se compose des risques et des problèmes qui se produisent dans le développement du système. C'est parce que les parties prenantes, utilisateurs ou les développeurs ont un manque de connaissances dans le processus du génie logiciel, en particulier dans la méthodologie d'ingénierie des exigences, modèles et techniques, ainsi que le support d'information nécessaire sur les exigences non fonctionnelles. Les parties prenantes, les utilisateurs et les équipes de développeurs doivent tirer partie de leurs connaissances et s'appuyer sur l'ingénierie du logiciel et l'ingénierie des exigences pour développer un nouveau système. Le manque de connaissances des utilisateurs dans le génie logiciel constitue un risque lié à la connaissance, etc. [Arm 05]. Aussi les auteurs ajoutent dans [Adi 13] que les risques dans les exigences non-fonctionnelles peuvent entraîner des risques de sécurité et des risques légaux.
- La volatilité des exigences se réfère à des ajouts, des suppressions et des modifications des besoins durant le processus de développement des systèmes [Kas 10]. Les risques de la volatilité des exigences est un problème qui se produit lorsque les conditions changent dans la relation entre les parties prenantes, les utilisateurs et les développeurs.
- Les risque de la documentation se produit quand les développeurs et les utilisateurs ne se servent plus des moyens et des outils de stockage appropriées pour documenter la progression du processus des exigences [Mea 05]. Il en résulte une perte d'attributs, de la difficulté à tracer, planifier et hiérarchiser des tâches dans les activités du processus d'analyse des besoins.

Des études ont effectivement mis en évidence l'échec du développement d'applications de taille importante dû à une définition des besoins « inadéquate » [Boe 81].

Les erreurs commises lors du processus d'ingénierie des besoins peuvent avoir de très lourdes conséquences pour les activités se trouvant en aval et donc pour le système logiciel résultant. La Figure 3.3 montre l'impact qu'une erreur faite lors du processus d'ingénierie des besoins peut avoir sur les activités de conception, d'implémentation et de test [Dav 93].

L'expérience a montré qu'une erreur de spécification aussi infime soit-elle détectée en cours d'exploitation, peut non seulement entraîner des surcoûts de correction considérables, mais aussi provoquer des dégâts dont les conséquences sont parfois irréversibles sinon dramatiques comme la perte de vie humaines dans les systèmes critiques.

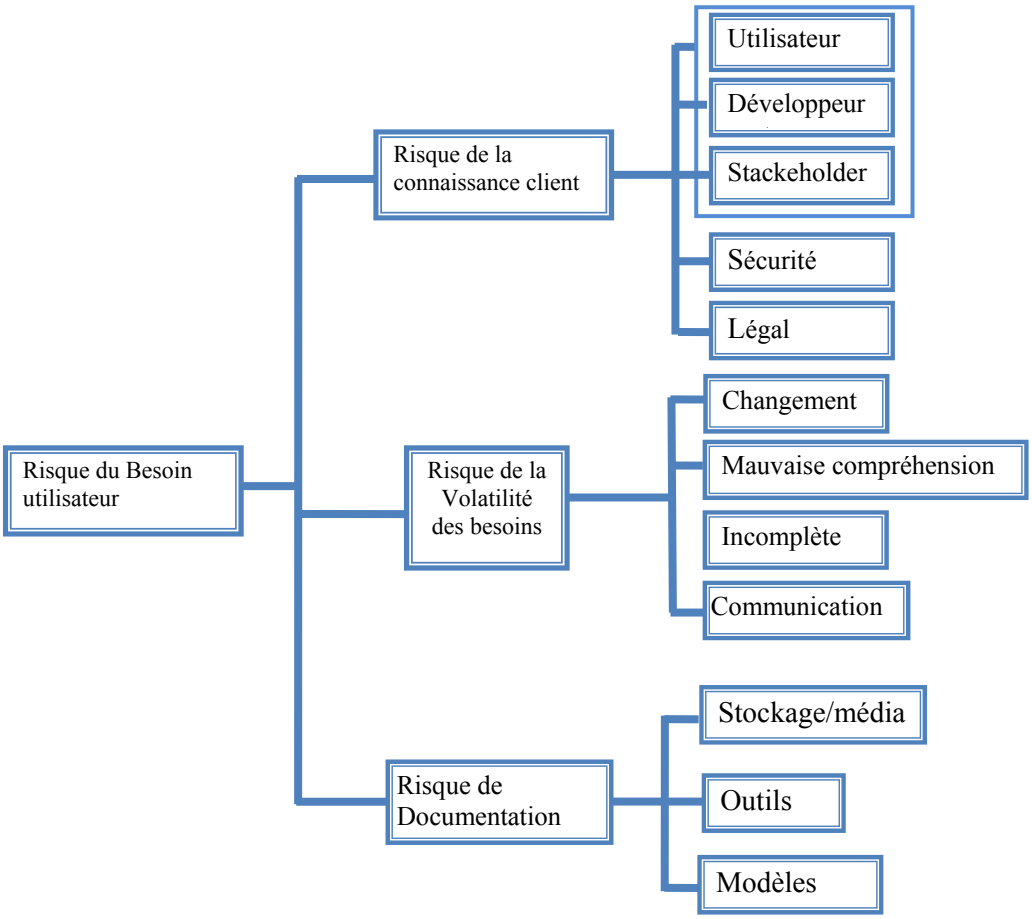


Figure 3.2 Classification des risques des besoins utilisateurs [Adi 13]

3.2 Les Approches pour l'Ingénierie des Besoins

La littérature propose plusieurs approches d'ingénierie des exigences. Chacune d'entre elles se concentre sur des activités spécifiques du processus d'ingénierie des exigences. Dans ce qui suit, nous présentons brièvement quelques approches :

- L'approche à base de scénarios [Sut 03], [Rol 98]

Dans ce type d'approche, les exigences sont décrites à l'aide de scénarios. Ces derniers sont des descriptions du monde réel qui sont exprimées à l'aide du langage naturel, de diagrammes, etc.

- L'approche GORE (Goal Oriented Requirements Engineering)

L'approche GORE se base sur la définition des exigences comme étant des buts qui peuvent être divisés et raffinés [Lam 09].

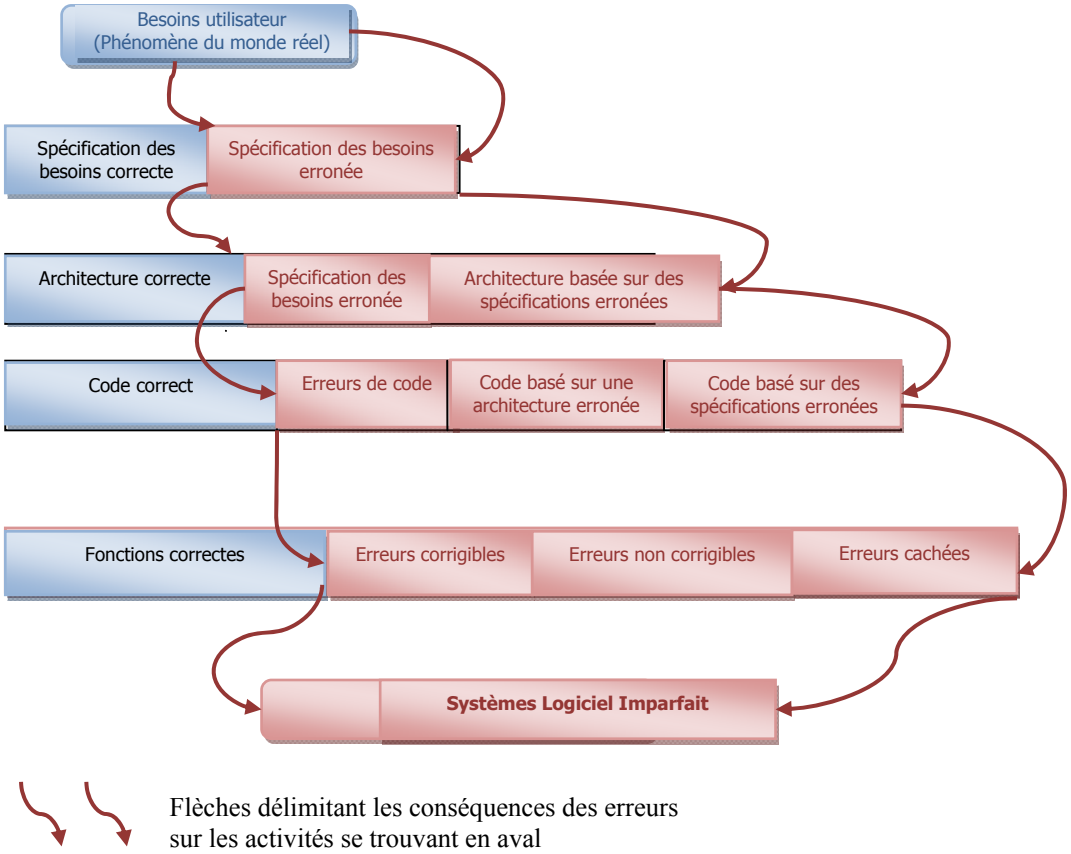


Figure 3.3 Effets des erreurs faites lors de l'ingénierie des besoins sur le reste du cycle de vie [Dav 93]

- L'approche orientée-aspects [Ras 08]

Cette approche reconnaît explicitement l'importance de bien identifier, de traiter et de séparer des préoccupations transversales qui ont été par ailleurs réparties dans des artefacts d'autres exigences (des cas d'utilisation, des modèles de buts, etc.).

- L'approche « Problem Frames » (schéma de problème) [Jac 01]

C'est une approche qui permet de structurer l'analyse des exigences d'un logiciel et de concevoir une solution logicielle. Elle aide à comprendre d'une part le contexte dans lequel réside le problème et d'autre part les aspects pertinents pour la conception d'une solution. Cette approche s'intéresse surtout aux exigences fonctionnelles.

Dans la dernière décennie, la popularité de l'approche GORE a augmenté et cela est dû au fait que son processus d'élaboration se termine là où la plupart des autres approches d'ingénierie des exigences débutent [Lam 02]. Les approches GORE se concentrent beaucoup sur les activités qui précèdent la formulation des exigences [Nus 00]. En effet, la plupart des autres approches d'ingénierie des exigences sont concentrées sur ce que le

logiciel doit faire et comment il doit le faire en traitant les exigences seulement en terme de processus et de données et en ne s'intéressant donc pas à la justification de ces exigences.

Par conséquent, il sera difficile par la suite de comprendre les exigences et de juger si elles capturent vraiment tous les besoins des parties prenantes.

Aussi, dans [Cas 02], les auteurs affirment que la tendance de la plupart des autres approches est d'abstraire les constructions de programmation (de bas niveau) au niveau des exigences plutôt que de pousser les abstractions des exigences jusqu'au niveau de la conception.

3.2.1 Approche dirigée par les buts : GORE

La naissance de l'approche GORE remonte à plus de deux décennies avec précisément le travail de Yue [Yue 87] qui était le premier à affirmer que la modélisation explicite des buts dans les modèles d'exigences peut fournir un critère pour la complétude des exigences. Dans [Dar 93, Lam 09], on trouve une définition du but comme un objectif que le système et son environnement doivent réaliser grâce à la coopération de différents agents (matériel, logiciel ou humain).

Un but placé sous la responsabilité d'un agent du système est appelé une exigence (*requirement*), tandis qu'un but placé sous la responsabilité d'un agent de l'environnement du système est appelé une attente (*expectation*). Les principales activités suivantes sont normalement présentes dans la plupart des méthodes basées sur l'approche GORE :

- l'élicitation des buts,
- le raffinement des buts,

Divers types d'analyse des buts et l'attribution de la responsabilité d'un but à un agent.

Nous présentons dans ce qui suit, une panoplie de méthodes adoptant le paradigme GORE.

3.2.2 L'Approche i*

Le framework iSTAR ou i*, acronyme de « Intentional STratégic Actor Relationships » [Yu 95, Yu 97], part du principe que les acteurs (les parties prenantes ou les entités actives) d'un système sont non seulement reliés entre eux par leurs actions ou l'information qu'ils échangent, mais aussi par les attributs intentionnels (les buts, les capacités, les croyances et les engagements) qui les caractérisent.

Le framework i* comporte deux modèles principaux : le modèle de dépendance stratégique et le modèle de raisonnement stratégique. Dans ces modèles i*, les acteurs sont décrits

dans leur contexte organisationnel et dépendent les uns des autres pour réaliser leurs buts, exécuter leurs tâches, ou disposer de ressources.

i* peut être utilisé tout au long des phases du processus d'ingénierie des exigences. Pendant les premières phases, le framework i* est utilisé pour faciliter l'analyse du domaine par des diagrammes qui permettent de représenter les acteurs du système, leurs buts et leurs relations. Les modèles i* développés à ce stade aident à comprendre pourquoi le futur système est nécessaire. Durant les dernières phases de l'ingénierie des exigences, les modèles i* sont utilisés pour proposer de nouvelles configurations du système qui seront évaluées en fonction de la façon dont elles répondent aux besoins fonctionnels et non-fonctionnels des utilisateurs. Pour cela, les softgoals (comme dans le NFR framework) sont utilisés comme critères de sélection pour choisir l'alternative du processus de configuration qui répond le mieux aux exigences non-fonctionnelles du système. En i*, il y a les liens de type « moyen-finalité» (means-ends en anglais) entre les buts pour spécifier les différentes alternatives pour réaliser un but. Les liens de décomposition quant à eux connectent un but/tâche avec ses composants (sous-tâches, softgoals, etc.). Un softgoal, un but ou une tâche peuvent être également liés à d'autres softgoals avec des liens de contribution, comme dans le NFR framework. Ces liens de contribution spécifient deux niveaux de contributions positifs (+ et ++) et négatifs (- et –) pour la satisfaction d'un softgoal, la réalisation d'un but, ou l'exécution d'une tâche. La figure 3.4 illustre un exemple.

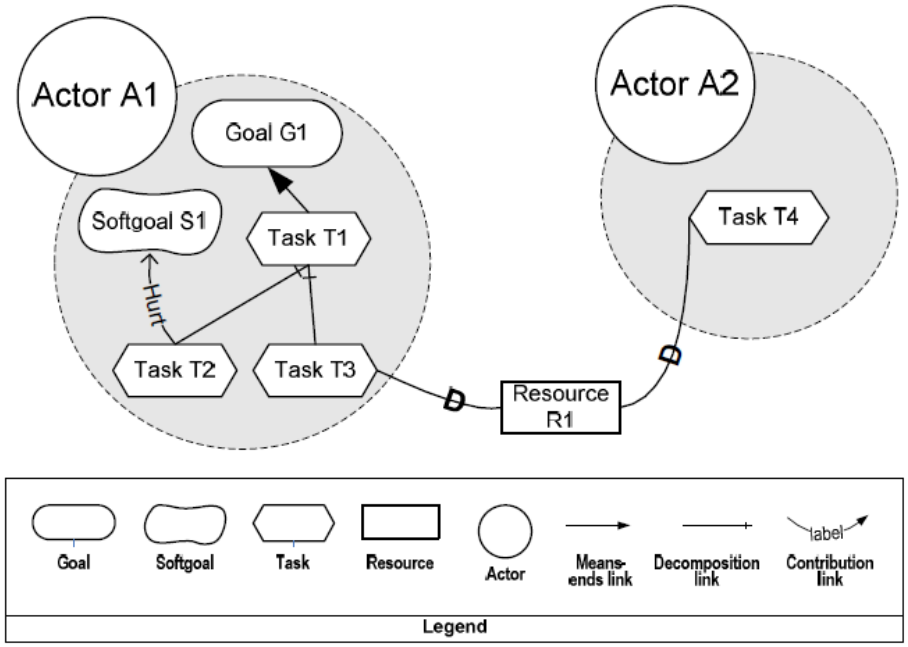


Figure 3.4 Exemple d'un modèle de buts illustrant ses concepts de base

Un langage, basé sur i* et nommé Goal-oriented Requirements Language (GRL), définit tous les concepts de i* (but, tâche, ressource, liens de contribution, etc.). GRL [Liu 01] est une partie de la notation des exigences utilisateur (URN) qui est une nouvelle recommandation de l'union internationale des télécommunications (ITU). URN fournit le premier standard des langages orientés-buts et il intègre aussi la notation UCM (Use Case Maps). TROPOS 3 [Cas 01] est une méthode d'ingénierie logicielle orientée-agent qui couvre tout le processus de développement logiciel. TROPOS se base sur i* afin de couvrir les toutes premières phases de l'analyse des exigences, permettant ainsi une meilleure compréhension :

1. de l'environnement dans lequel le logiciel doit fonctionner ;
2. des types d'interactions qui devraient se produire entre les logiciels et les agents humains.

TROPOS adopte le framework i* afin d'exprimer les concepts d'acteur, de buts et de leurs interdépendances.

3.2.3 Approche KOAS

KAOS, acronyme de « Keep All Objectives Satisfied » [Dar 93, Lam 09], est une méthode d'ingénierie des exigences qui résulte des travaux de recherche menés à l'Université de Louvain, en collaboration avec l'Université d'Oregon. Cette méthode permet aux analystes de construire des modèles d'exigences et de produire des documents à partir de ces modèles. KAOS a distingué les buts des propriétés du domaine qui sont des déclarations descriptives sur l'environnement telles que des lois physiques ou des normes organisationnelles.

Le modèle des exigences KAOS se compose de cinq sous-modèles fortement liés par des règles de cohérence :

Le modèle central : c'est le modèle de buts qui décrit les buts du système et de son environnement. Ce modèle est organisé dans une hiérarchie obtenue grâce au raffinement de buts de plus haut niveau (les buts stratégiques) vers des buts de bas niveau (les exigences).

Le modèle objet : Il permet de décrire le vocabulaire du domaine. Il est représenté par un diagramme de classes UML.

Le modèle des responsabilités : Il permet d'assigner les exigences (les buts feuilles) aux différents agents. Ces agents appartiennent au système à construire (agents internes) ou à son environnement (agents externes).

Le modèle des opérations : Il représente les opérations du système en termes de caractéristiques individuelles et leurs liaisons avec les modèles de buts (liens

d'opérationnalisation des exigences), objet (liens entrée-sortie) et responsabilités (liens d'exécution).

Le modèle des comportements : Il résume tous les comportements que les agents doivent accomplir pour satisfaire les exigences.

La Figure 3.5 donne un aperçu des quatre premiers sous-modèles KAOS ainsi que des principaux éléments qui les constituent. Le modèle des exigences KAOS peut être vu donc comme une sorte de puzzle visant à assembler les différentes pièces identifiées au fur et à mesure que les sources d'information sont exploitées. Le modèle de buts joue un rôle central dans la méthode KAOS. Par exemple, il est possible de dériver certains modèles KAOS (le modèle des opérations, le modèle objet, etc.) à partir du modèle de buts d'une façon systématique [Lam 09]. De plus, le modèle de buts permet de supporter très tôt différentes formes d'analyse des exigences telles que l'analyse de risques, l'analyse de conflits, ou l'évaluation des alternatives.

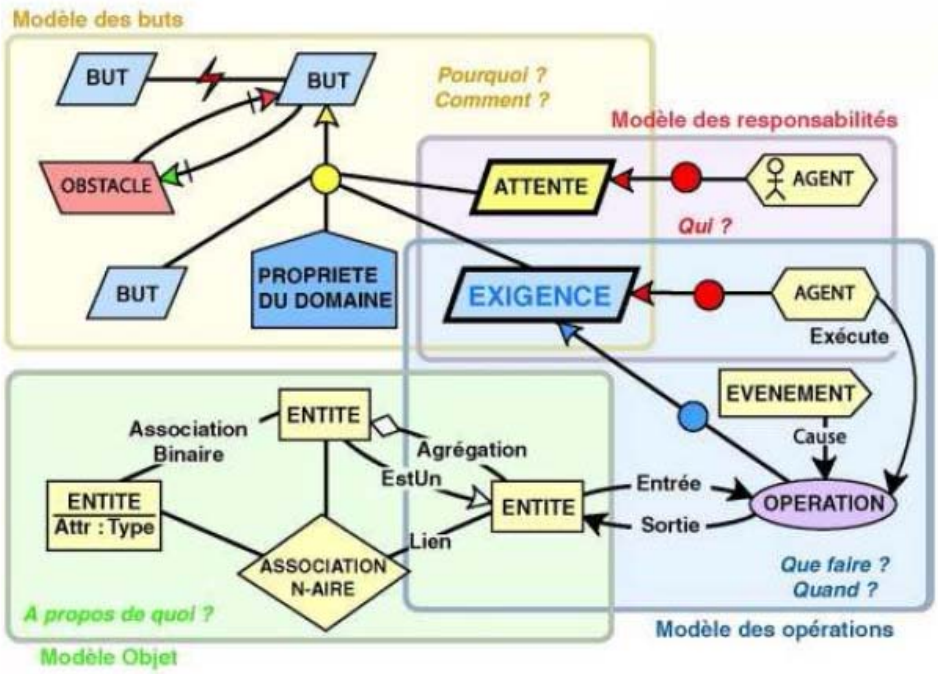


Figure 3.5 Un Aperçu du Modèle KAOS

Comme dans la plupart des méthodes GORE, les buts dans KAOS peuvent être raffinés selon des raffinements ET ou OU. Un raffinement ET implique que la conjonction des sous-buts est une condition suffisante pour réaliser le but parent.

Un raffinement OU associe un but à des sous-buts alternatifs pour lesquels l'accomplissement du but de plus haut niveau exige l'accomplissement d'au moins un de ses sous-buts. KAOS offre un ensemble de patrons de raffinement [Lam 09] qui décomposent les buts.

3.3 La discipline Ingénierie des Exigences du RUP

La discipline ingénierie des exigences occupe une place importante dans le processus de développement du RUP. En plus d'être l'élément de base qui sert les autres disciplines telles que l'analyse et la conception, les exigences permettent également d'estimer le coût et l'effort du développement du système à produire. La discipline de la gestion des exigences contient l'enchaînement des activités qui permettent d'analyser le problème à résoudre, de définir le système, de comprendre les besoins des utilisateurs, de documenter les exigences logicielles et de gérer les changements.

Les objectifs de la discipline de la gestion des exigences se résument donc dans les points suivants [Pas 06] :

- Établir un accord entre les intervenants sur les objectifs du système à développer.
- Fournir aux développeurs du système une meilleure compréhension des exigences logicielles.
- Définir les limites du système à développer.
- Fournir un plan initial des itérations à réaliser.
- Fournir des estimations initiales des coûts, des échéanciers pour développer le système.
- Définir les interfaces graphiques en se basant sur les besoins des utilisateurs.

La Figure 3.6 illustre l'enchaînement des tâches pour la discipline gestion des exigences.

Chaque tâche est présentée par un enchaînement d'activités cohérentes les unes par rapport aux autres.

Le processus réel employé par la discipline requise est largement axé sur les cas d'utilisation, qui sont supposés être un milieu approprié pour la communication des exigences fonctionnelles entre les parties prenantes et les développeurs. Pour cet effet, les Cas d'utilisation ainsi que toutes les autres spécifications et des documents traités dans cette discipline devraient également être créés dans la langue des clients.

Dans les sections suivantes, chaque activité impliquée dans la discipline Exigences est brièvement expliquée. En outre, étant donné que chaque activité peut introduire d'autres tâches et des artefacts, des diagrammes montrant plusieurs des relations entre les tâches et artefacts seront affichés.

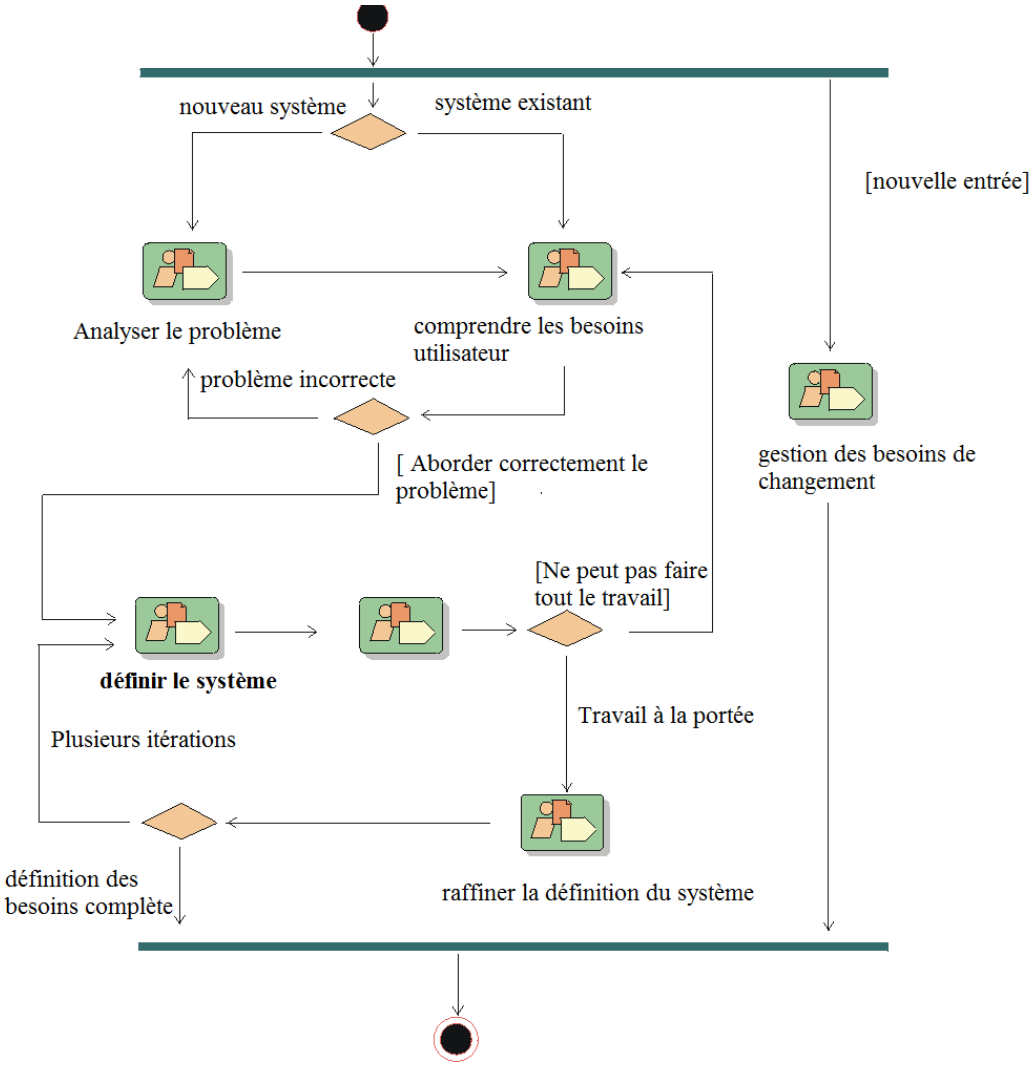


Figure 3.6 Enchaînement des activités de la discipline Exigences [Pas 06]

3.3.1 L'activité : Analyse Problème

L'activité "Analyse problème», décrit les tâches initiales à effectuer si un nouveau système doit être développé. L'aspect le plus essentiel de cette activité est d'identifier les parties prenantes clé ainsi que les exigences clés pour le projet (Figure 3.7).

Afin de cerner les limites du système, une partie de la tâche ″Développer Vision″ est de parvenir à un accord sur quelques problèmes réels à résoudre. Basé sur cette information, le premier artéfact le ″Document Vision″ pourra être rédigé. Ce document est destiné à décrire la vision globale du projet ainsi que pour documenter l'information sur les intervenants clés et les problèmes identifiés à ce jour.

Pour éviter tout malentendu entre les parties prenantes dans le projet et aussi faciliter la compréhension et une vision commune, la tâche "Capturer vocabulaire commun″ recueille les terminologies les plus importantes utilisé dans le domaine du problème. Cette

information est ensuite recueillie et structuré dans le glossaire, qui est maintenue et améliorée tout au long du reste de processus de développement.

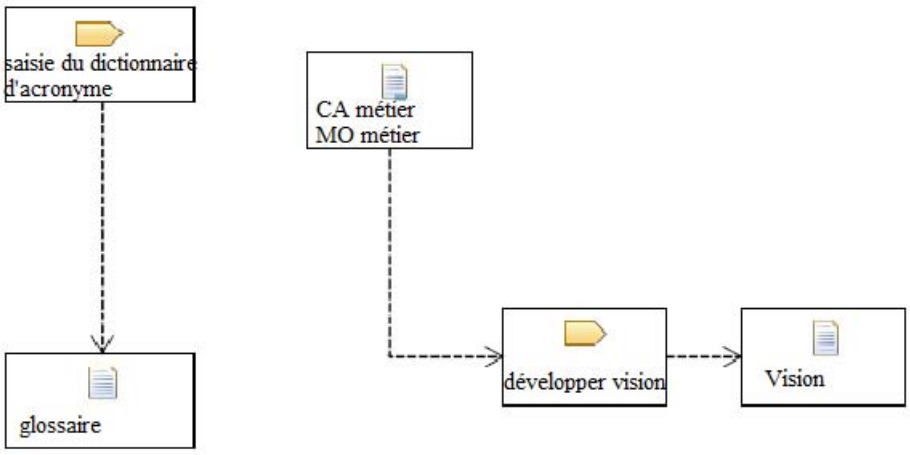


Figure 3.7 Tâches et artéfacts impliqués dans l'activité : Analyse du Problème

3.3.2 L'activité : Comprendre les Besoins Utilisateurs

Après avoir identifié les parties prenantes dans l'activité précédente, des exigences plus détaillées peuvent désormais être collectées. La tâche ″Eliciter demandes intervenants″ inclut le processus d'interview des clients, l'évaluation des questionnaires et d'autres techniques visant à la collecte des besoins plus spécifiques. Une des techniques mises en évidence par le RUP est ″Storyboarding″ qui présente un ensemble de scénarios ou comportement du système créés et raffinés par les utilisateurs.

Afin d'être en mesure de comprendre les intentions derrière les besoins individuels formulé à ce stade et les étapes ultérieurs, il sera désormais utile d'avoir une bonne compréhension du contexte du système, comme indiquée par la ″Modélisation métier″.

En parallèle à la collecte des besoins fonctionnels, les besoins fonctionnels devraient désormais être identifiés. Contrairement aux exigences fonctionnelles qui sont incorporées principalement dans les cas d'utilisation, les exigences non fonctionnelles sont recueillies et structurées dans un artéfact à part appelé SS ″spécification supplémentaire″ (Figure 3.8).

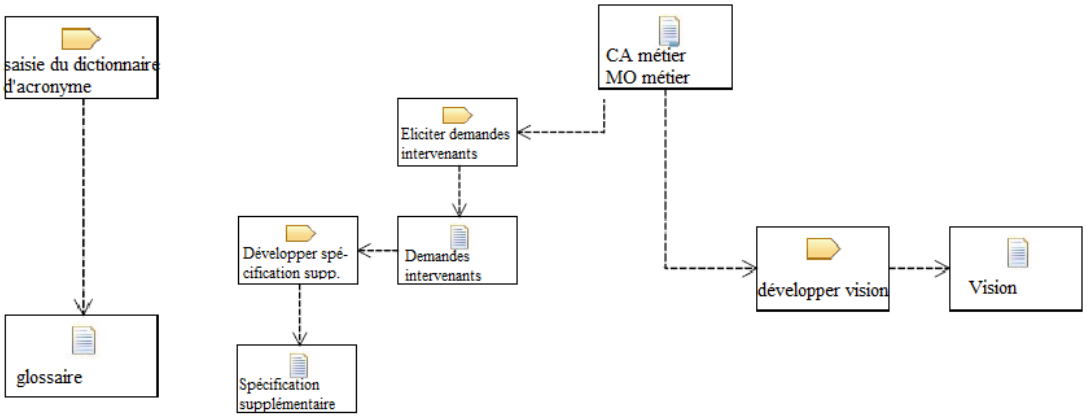


Figure 3.8 Tâches et artéfacts impliqués dans l'activité : Comprendre les Besoins des Utilisateurs

3.3.3 L'activité : Définir le Système

La définition de la vision du système à construire et l'identification des intervenants et des exigences clés ont été élaborées par les deux activités précédentes. Ces informations peuvent être analysées plus amplement. La tâche la plus importante de l'activité ″Définir le système″ est donc : Trouver des acteurs, et des cas d'utilisation, qui visent à identifier et raffiner le système en se basant sur les informations recueillies.

Les cas d'utilisation et les acteurs identifiés doivent ensuite être expliqués. Bien que la définition précise soit élaborée dans des activités ultérieures, chaque cas d'utilisation et acteur doivent être accompagnés d'une brève description, compréhensible par le client. Dans le cas des acteurs, les responsabilités individuelles doivent être décrites de même que la valeur fournie par le système pour cet acteur spécifique. Dans les cas d'utilisation, une description générale ainsi qu'un grand nombre d'événements doivent être spécifiés.

La figure 3.9 montre les tâches et les artéfacts impliqués dans la définition d'un système.

3.3.4 L'activité : Gestion de la Portée du Système

Le résultat des activités précédentes est l'identification d'un nombre important de cas d'utilisation. L'objectif l'activité « Gestion de la portée du système » est de structurer le modèle des cas d'utilisation et la priorisation des cas d'utilisation individuels.

L'ordre des priorités est abordé par la tâche "Prioriser cas d'utilisation" en s'appuyant sur plusieurs critères. En plus de la prise en compte de la valeur de chaque cas d'utilisation fourni aux parties prenantes, aussi bien que le budget et le temps prévu. Cette tâche utilise deux artéfacts provenant d'autres disciplines effectuées en parallèle (« Gestion de projet » et «analyse et conception »).

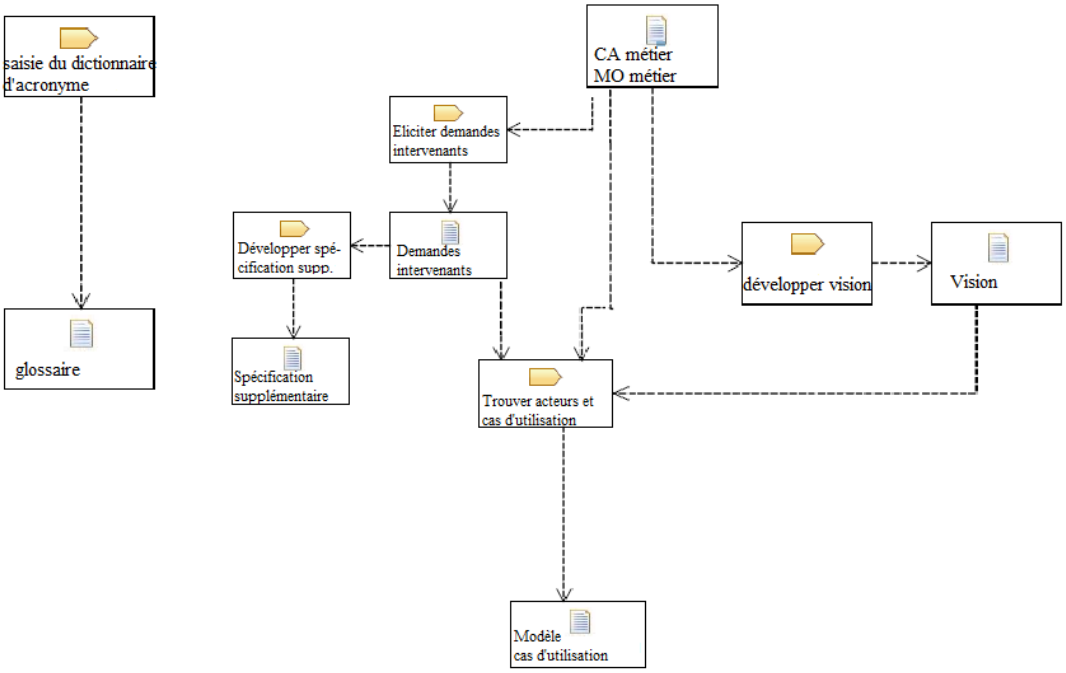


Figure 3.9 Tâches et artéfacts impliqués dans l'activité : Définition du système

La discipline "Gestion de Project" prévoit la ″Liste des risques", un artéfact qui documente les événements qui pourraient mener à un résultat négatif du projet.

Sur la base de ces informations, les cas d'utilisation qui posent un plus grand risque sur le succès du développement peuvent avoir une priorité à être mis en œuvre au début, tandis que les cas d'utilisation à faible risque seront reportés à une étape ultérieure.

Le deuxième artéfact utilisé est le «Document d'architecture logicielle», créé par la discipline «Analyse et Conception". Ce document décrit une première ébauche de l'architecture logicielle facilitant la mise en œuvre du système. Le fait que l'architecture choisie puisse impliquer des cas d'utilisation ayant un impact plus important sur l'architecture que d'autres, une hiérarchisation de priorité devrait être prise en considération vis-à-vis des cas d'utilisation individuels.

La figure 3.10 montre les tâches et les artéfacts impliqués l'activité « Gestion de la portée du système ».

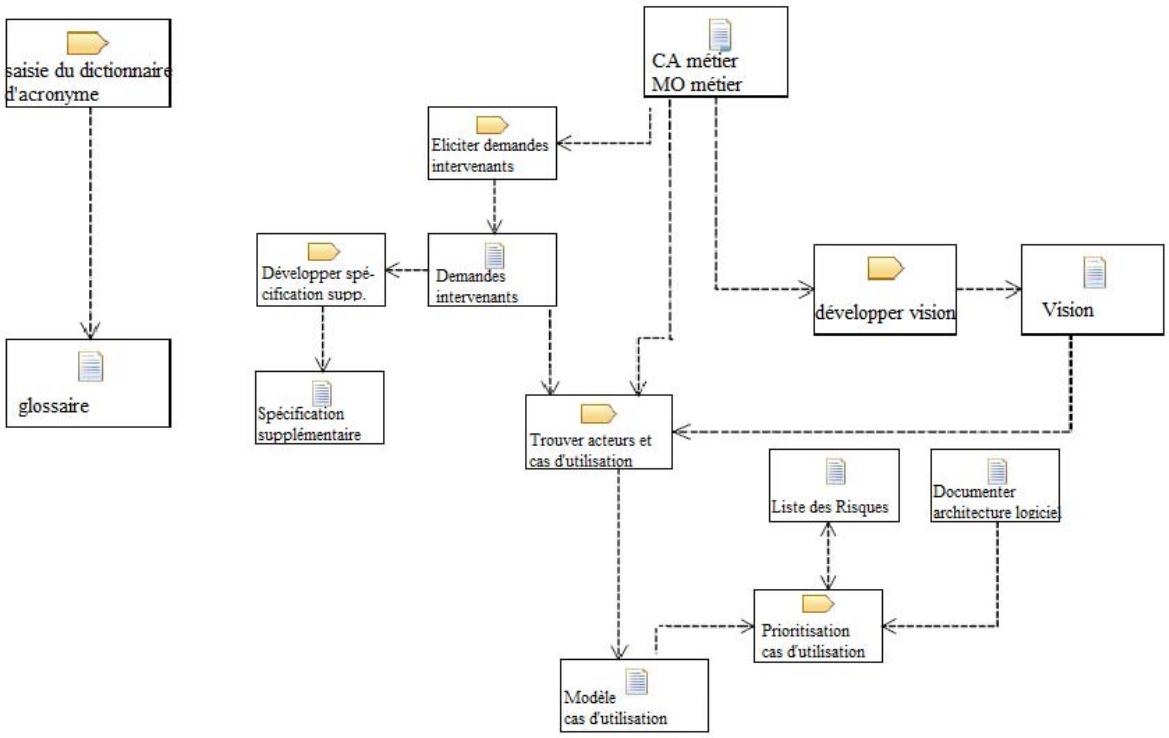


Figure 3.10 Tâches et artéfacts impliqués dans l'activité : Gestion de la portée du Système

3.3.5 L'activité : Raffiner la Définition du Système

A ce stade, la plupart des exigences ont été capturées et les premiers cas d'utilisation et les spécifications ont été créés. Une partie de l'activité "Raffiner la définition du système" est désormais de favoriser la compréhension de la portée du projet reflétée par l'ensemble des fonctionnalités prioritaire donné par l'activité précédente. D'autre part, les activités comprennent des artéfacts existants détaillés, et donc destiné à compléter la première étape de l'ingénierie des exigences.

Chacun des cas d'utilisation défini dans l'activité « Définir le système» et d'autres activités, sont maintenant détaillés profondément. Cela peut se faire en fournissant des descriptions textuelle détaillées ainsi que en créant des diagrammes d'états ou des diagrammes d'activité portant le flux des événements détaillé pour chaque cas d'utilisation. Surtout dans le cas d'interfaces utilisateur où les prototypes peuvent être créés pour rassembler des feedbacks et de nouvelles propositions des utilisateurs et intervenants.

Analogue aux exigences fonctionnelles exprimées par les cas d'utilisation, les exigences non-fonctionnelles contenues dans la spécification supplémentaire devraient être encore spécifiés et complétées.

Eventuellement, une spécification des exigences logicielles (SRS) peuvent être créés à ce stade du processus. Le SRS représente un document unique ou une collection d'artéfacts consolidant et décrivant l'ensemble complet d'exigences capturées. Le SRS peut être rédigé sous forme de texte uniquement, ainsi que comme une combinaison de texte et de cas d'utilisation. Cependant, les documents ne sont pas définitifs, mais font l'objet de mises à jour tout au long du cycle de vie du projet, afin de toujours tenir compte de l'ensemble le plus courant des exigences.

La figure 3.11 montre les tâches et les artéfacts impliqués dans l'activité « Raffinement de la définition du système ».

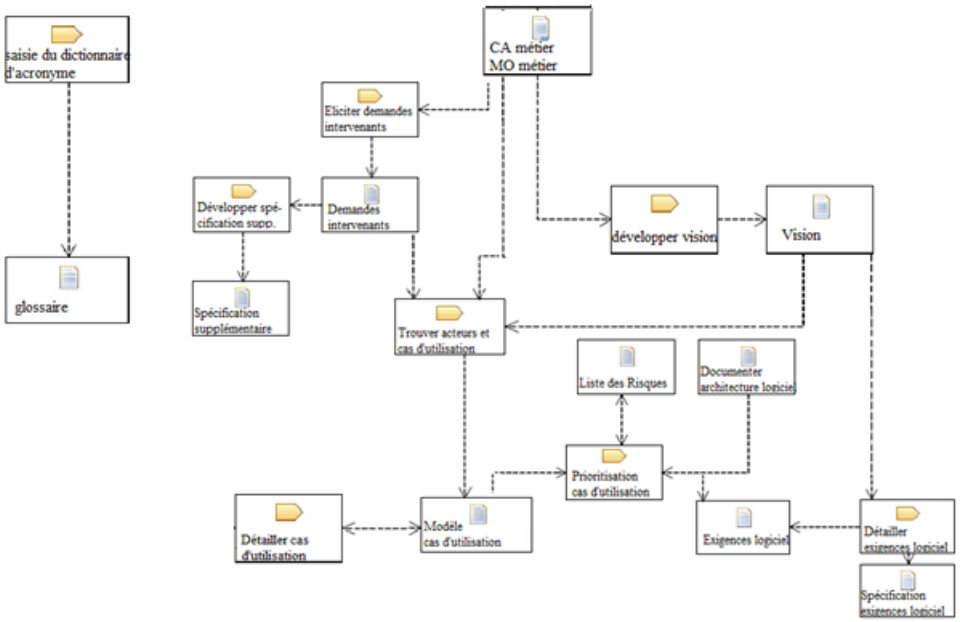


Figure 3.11 Tâches et artéfacts impliqués dans l'activité : Raffinement de la définition du système

3.3.6 L'activité : Gestion du Changement des Besoins

Comme souligné auparavant, RUP met l'accent sur le fait que les exigences sont susceptibles de changer tout au long du cycle de développement. La gestion de ces nouvelles exigences est le but de l'activité ″Gestion de l'évolution des besoins".

Un aspect de cette activité est donc d'évaluer l'impact de l'évolution des besoins sur les autres exigences et de les incorporer dans le système à développer.

Un autre aspect de cette activité est d'améliorer et de maintenir les différents modèles créés, ceci est assuré par la tâche "le modèle des cas d'utilisation".

En outre, des réunions régulières avec les parties prenantes devrait être tenues dans lesquelles les exigences sont revues.

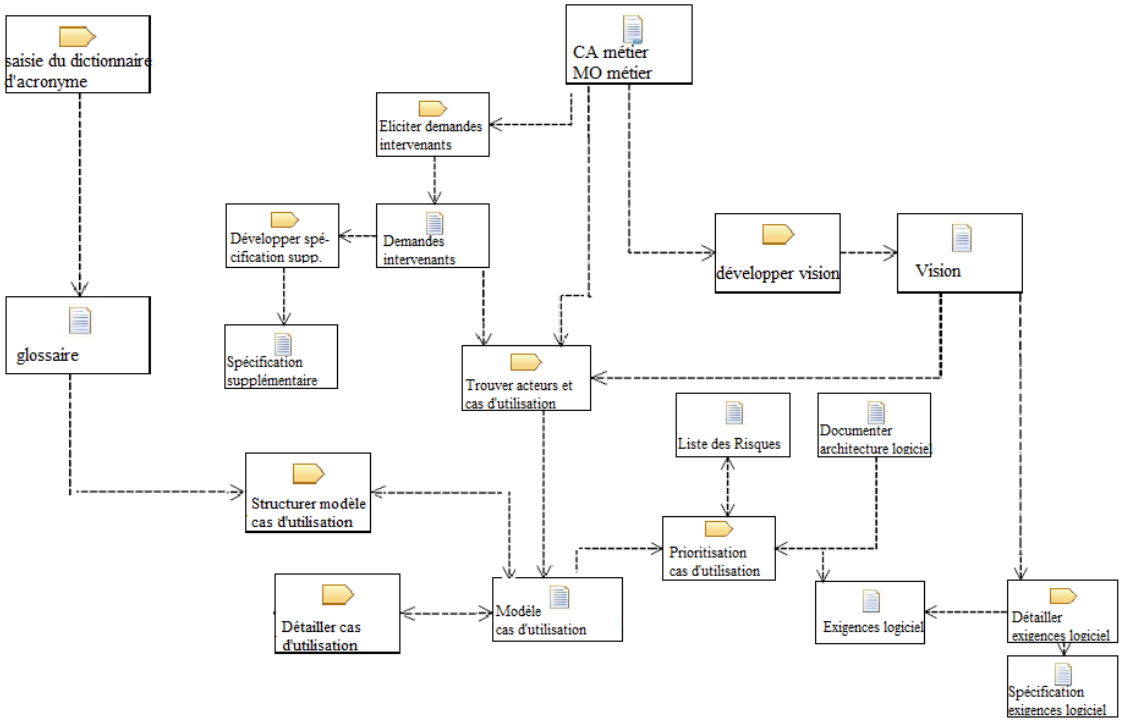


Figure 3.12 Tâches et artéfacts impliqués dans l'activité : Gestion du Changement des Besoins

3.4. Ingénierie des Besoins et Evolutions anticipées

3.4.1 L'extraction des Besoins

Découvrir les exigences actuelles d'un système représente déjà une tâche complexe. De ce fait, s'intéresser aux besoins futurs est une tâche au moins aussi difficile mais souvent très difficile. De plus il est impossible de savoir avec certitude si un événement futur va vraiment avoir lieu. Néanmoins, la compréhension des évolutions futures est globalement, et relativement, moins difficile que la compréhension détaillée du système logiciel (ce qu'il faut entreprendre) lorsqu'elle est entamée la première fois.

3.4.1.2 Représentation du Futur

3.4.1.2.1 Notion d'évènement futur

Un événement futur est un événement qui devrait avoir lieu à l'avenir [Pim 11]. En ce qui concerne l'analyse du problème et d'après Kotonya et Sommerville [Kot 98], il existe quatre dimensions liées à l'élicitation des exigences :

1. Domaine d'application,
2. Problème à résoudre,
3. Contexte d'affaires et
4. Besoins et contraintes des parties prenantes.

Si nous voulons obtenir des exigences portant sur des questions futures, la prise en compte de ces quatre dimensions est nécessaire dans ce futur [Pim 11].

Plusieurs techniques [Gor 04], [Pim 11] et méthodes permettent la découverte rationnelle des futurs ou avenirs possibles. Ces futurs peuvent être juste un futur spécifique attendu, par exemple une date précise ou peut être plusieurs différents avenirs possibles. Ils sont souvent énoncés sous forme de diagrammes, des descriptions textuelles [Gle 72] ou des représentations mathématiques [Bla 72]. Les méthodes prospectives (voir Table 3.3) peuvent être classées en tant que qualitative ou quantitative, et ils peuvent étudier aussi bien le futur qu'avoir d'autres utilisations comme c'est le cas en économétrie et scénarios [Pim 11].

3.4.1.2.2 Représentation du futur

Une représentation de l'avenir ou du futur peut être soit intentionnellement ou accidentellement créée, sous une forme formelle ou informelle [Lov 96]. Par conséquent, elle peut occuper n'importe quelle position sur l'axe de la figure 3.13. Les meilleurs résultats sont obtenus si un modèle du futur est formel et créé intentionnellement [Pim 11]. Cependant, il n'est pas possible que tous les projets disposent de ressources suffisantes pour créer un tel modèle. En outre, pour certains systèmes, ceci est difficile. Dans de tel cas, l'ingénieur des exigences peut collecter des indices sur le futur, en utilisant des techniques d'élicitation préconisées, dont [Eck 96] :

- L'écoute des commentaires des parties prenantes pendant les séances « groupwork » l'examen de l'environnement réglementaire, et
- L'analyse des plans des clients.

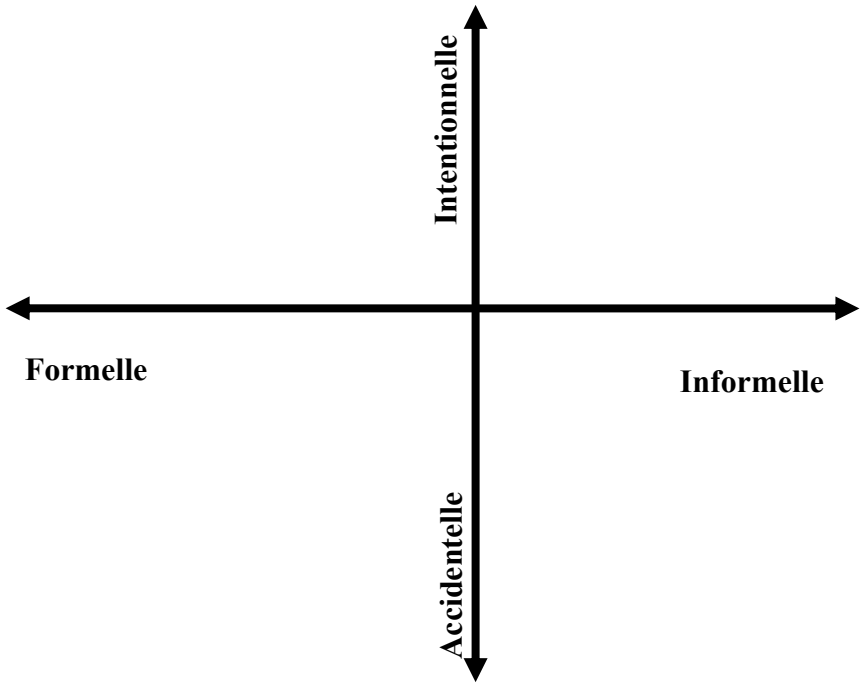


Figure 3.13 Axes de caractérisation d'une représentation du futur

3.4.1.2.3 Les méthodes futuristes

Une méthode de prospective est un moyen de création d'une représentation du futur [Pim 11]. Gordon et Glenn organisent dans [Gor 04] les méthodes ou les groupes de méthodes de la prospective, par rapport à ce qu'on souhaite atteindre :

a) Collection des jugements des experts

b) Prévision temporelle et autres mesures quantitatives

c) Comprendre les liens entre les évènements, tendances et actions

d) Déterminer un plan d'action en présence de l'incertitude;

e) Représentations alternatives des futurs plausibles

f) Parvenir à un accord si l'avenir s'améliore;

g) Tracer des changements et suppositions

h) Déterminer la stabilité d'un système

Catégorie	Méthode
Collection des jugements des experts	- Delphi - Recueillir des jugements d'experts - Méthodes participatives
Prévision temporelle et autres mesures quantitatives	- Prévisions économétrie - Analyse de régression - Analyse des tendances d'impact - Analyse structurelle
Comprendre les liens entre les évènements, tendances et actions	- Dynamique des Systèmes - Modélisation Agent - Analyse des tendances d'impact - Etude d'impact transversale - Arbres de Pertinence - Roue des futurs - Modélisation de simulation - Perspectives multiples - Analyse causale en couches - Relaxation des anomalies du champ
Représentations alternatives des futurs plausibles	- Scénarios - Roue des futurs - Simulation et jeux - Modélisation Agent

Table 3.3 Méthodes Futur classées selon leur utilisation

Quatre de ces huit buts sont étroitement liés à l'élicitation des besoins.

Recueillir des jugements. C'est la catégorie des méthodes qui peuvent être utilisées quand il est nécessaire de réduire le degré de l'incertitude d'un projet.

Prévision temporelles et autres mesures quantitatives. Catégorie qui peut être utilisée pour résoudre les problèmes d'évolutivité et de sécurité. Par exemple, en estimant la charge future d'un système.

Comprendre les liens entre les événements, des tendances et les actions. Catégorie qui peut être utilisée lorsque cela est nécessaire pour comprendre comment le changement d'une exigence aura un impact sur les autres.

Représentations alternatives des futurs plausibles. C'est la catégorie des méthodes qui peuvent être utilisées quand il est nécessaire de comprendre un scénario d'utilisation futur, permettant l'anticipation des changements requis pour supporter ce scénario.

Il ya des méthodes de prospection liées aux recherches en génie logiciel; telles que, Delphi [Boe 81], la dynamique des systèmes [Mao 07], Modélisation Agent [Tes 00] et les jeux de simulation [Boi 03]. Quelques méthodes de prospection sont également utilisées pour l'élicitation des besoins, mais pas du point de vue de l'étude de l'avenir. Comme, par exemple, les méthodes participatives et les scénarios [Pim 11].

3.4.1.3 La Méthode Futures Wheel

La Roue des futurs ″ Futures wheel″ est une méthode de brainstorming structurée utilisée pour organiser la réflexion sur les événements futurs, les questions, les tendances et la stratégie [Gle 72]. C'est une méthode prospective qui fournit un modèle de l'avenir sur la base des conséquences d'un événement ou d'une tendance. La méthode "Futures Wheel" est souvent utilisée pour :

- Réfléchir à des impacts possibles des tendances actuelles ou des événements futurs potentiels;
- Organiser les pensées sur des événements ou tendances futurs;
- Créer des prévisions dans des scénarios alternatifs;
- Montrer les interrelations complexes;
- Afficher d'autres recherches sur le futur;
- Développer des concepts multiples; favoriser une perspective à terme et aider au raisonnement.

C'est une méthode subjective et qualitative qui s'appuie sur l'expérience et les connaissances des participants. Sa faible complexité permet son utilisation sans nécessiter qu'une formation spécialisée soit menée. Néanmoins, elle nécessite une compréhension profonde du domaine du problème en cours d'analyse, de sorte que le modèle futur généré puisse être aussi précis que possible. Par conséquent, il est important qu'il y ait une forte implication des représentants des clients ou des experts du domaine lors de la génération des modèles.

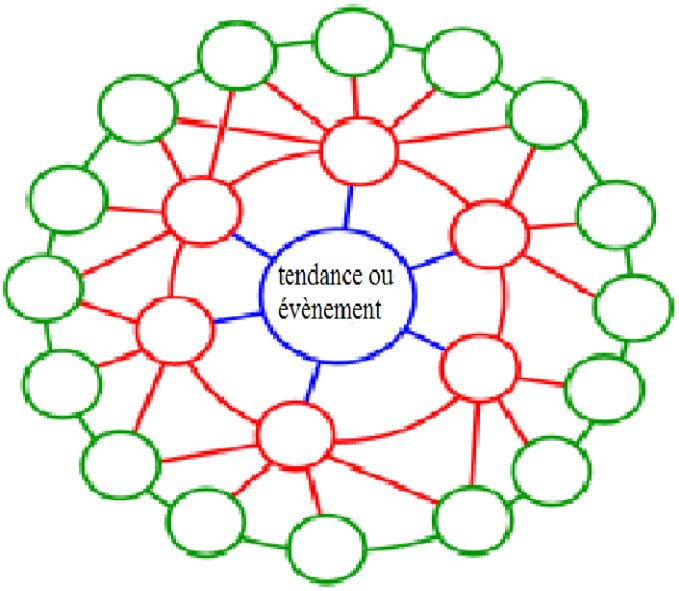


Figure 3.14 « «Futures wheel » comme décrite par Jerome C. Glenn 1972

La méthode elle-même se compose de deux étapes [Pim 11].

La première étape consiste à identifier des tendances ou des événements qui sont susceptibles de se produire dans un proche avenir et qui sont liés au domaine du problème. Une tendance est quelque chose qui a déjà commencé et est de plus en plus forte. Un événement futur est simplement quelque chose qui devrait se produire.

La deuxième étape consiste à affiner le cas par l'ajout des conséquences de certains événements. Pour chaque événement, nous demandons « Quels sont les impacts ou conséquences, de cet événement? » Ensuite, pour chaque conséquence, identifier les conséquences secondaires : à savoir, les conséquences des conséquences, les conséquences tertiaires, et ainsi de suite.

Roue des Futurs de base : Comment faire?

Un groupe de participants décide de réfléchir à une tendance, une idée, un évènement futur ou d'une valeur. Le sujet est écrit au milieu d'un morceau de papier, un tableau de conférence, tableau noir, ou sur un transparent de rétroprojecteur (Figure 3.15).

Figure 3.15 Exemple d'un évènement à étudier selon le principe de la roue des futurs

Ensuite, le chef de la session brainstorming dessine un ovale autour de la question et demande au groupe de dire ce qui se passe forcément avec cet article.

Puis, le chef de la session brainstorming dessine un ovale autour de la question et demande au groupe de dire ce qui se passe forcément avec cet élément. Comme les impacts ou les conséquences sont proposés par le groupe, le chef dessine des rayons courts sortants de l'ovale central et écrit ces impacts à la fin de chaque rayon. Des ovales sont dessinés autour de chacun des impacts primaires formant le premier anneau de la roue (Figure 3.16).

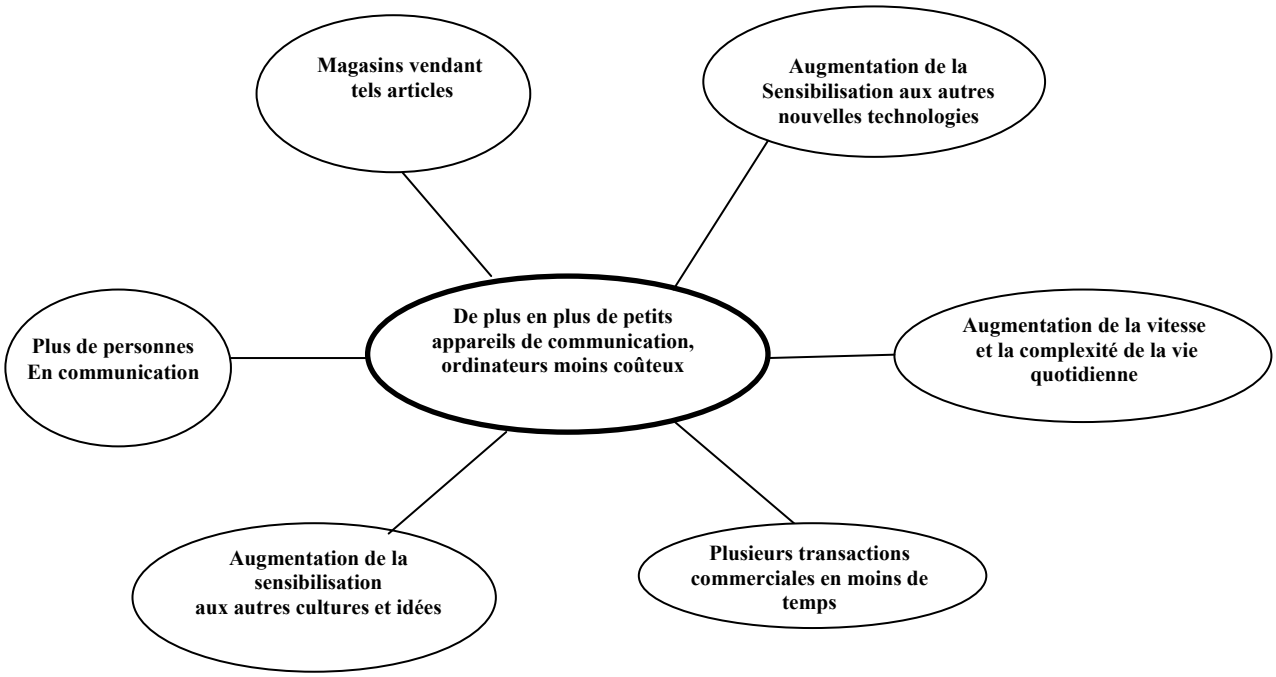


Figure 3.16 Exemple d'impact primaire d'une tendance

Ensuite, le chef demande au groupe d'oublier l'élément d'origine dans le milieu de la roue des futurs et de donner les impacts les plus probables pour chacun des principaux impacts du premier anneau des conséquences primaires (voir Figure 3.17).

Ainsi de suite, les participants listent des conséquences de deuxième, troisième et quatrième ordre avec une éventuelle évaluation. Après, lorsque le groupe estime que sa réflexion est représentée sur la roue, il peut évaluer et modifier la roue pour être plus «réaliste». Alternativement, les impacts d'un événement ou tendance peuvent être traitées plus lentement et délibérément en acceptant les critiques avant d'entrer quoi que ce soit sur la roue. Dans cette approche, le groupe discute de la plausibilité de chaque impact. Si un impact est jugée plausible par tous, alors il est introduit, sinon, il ne l'est pas.

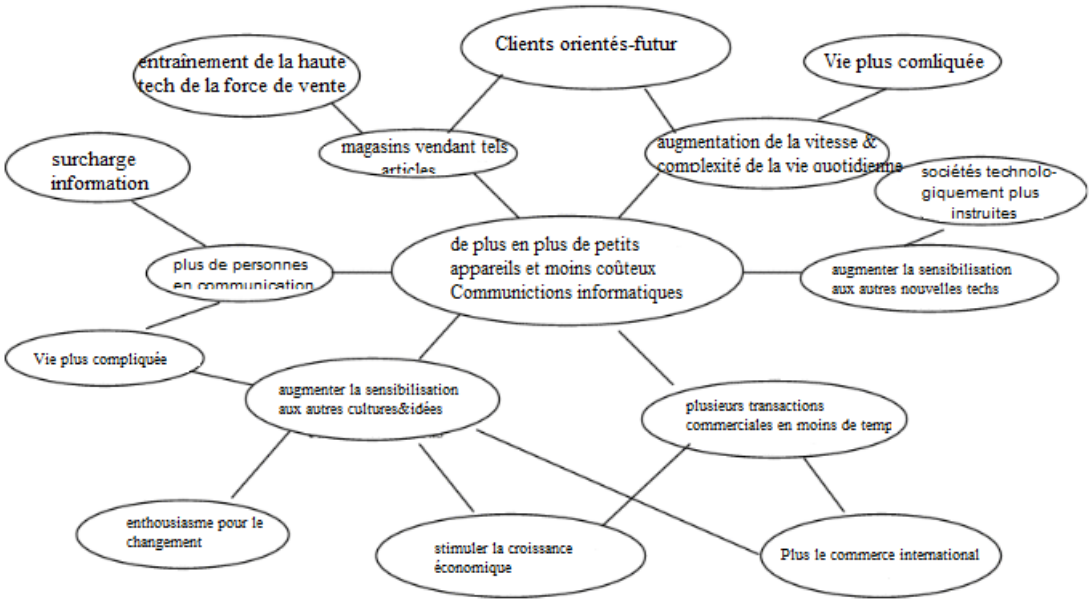


Figure 3.17 Exemple d'impact primaire et secondaire d'une tendance

3.4.2 Modélisation des Evolutions Futures : Cas de Changement

Un cas de changement [Eck 96] permet d'identifier et intégrer les changements anticipés dans un design pour améliorer sa robustesse à long terme. Un cas de changement peut décrire aussi bien les nouveaux besoins d'un système que les modifications des besoins existants. Ce sont des formalismes simples qui décrivent les changements futurs pouvant altérer un système.

Définition d'un cas de changement

Un cas de changement consiste en:

(a) un cas d'utilisation, avec scénario un nouveau ou révisé, dérivé d'un changement d'un besoin, et

(b) un ensemble de cas d'utilisation existants, qui ont besoin d'être changés pour rester conforme aux exigences modifiées.

Motivation au changement dans un système

Le changement de système est parfois une nécessité à cause d'une faible correspondance initiale entre le système et son contexte. Cependant, même si un système est bien apparié avec son contexte, le changement est inévitable.

- ***Changements de marché***. Une entreprise peut réussir et la base de ses clients aura tendance à croître et devenir plus exigeante.
- ***Changements des exigences métier***. Les organisations changent constamment de politiques et procédures pour faire face aux changements dans leur environnement. Par exemple, une nouvelle politique peut être établie pour vérifier l'état du compte d'un client avant l'expédition de marchandises pour répondre à une commande. Ce changement de politique aura une incidence sur le système de traitement des commandes de l'organisation.
- ***Changements du processus opérationnel***. Les informations ou des actions obligatoires sont redéfinies en raison des modifications apportées au processus opérationnel pour des raisons internes ou économiques. Par exemple, l'élimination de la saisie des données, par des agents dédiés, dans un processus de commande, peut exiger du système de valider les données brutes qui y entrent pour accroître ses capacités.
- ***Changements législatif ou réglementaire***. Des actions opérationnelles peuvent avoir besoin d'être changées et de nouvelles informations doivent être acquises face à un changement législatif ou réglementaire. Les opérations antérieures peuvent devenir interdites. Par exemple, une récente réglementation du Conseil des Normes de Comptabilité Financières exige des entreprises de tenir compte de la pension retraite et des prestations médicales, ceci nécessite que le système informatique concerné traite des informations qui n'étaient pas nécessaire auparavant.
- ***Imagination des utilisateurs***. Comme les utilisateurs deviennent plus familiers avec le système qu'ils utilisent, ils découvrent inévitablement de nouvelles façons qui pourraient mieux répondre à leurs besoins. Les utilisateurs pourront demander des améliorations qui les rendent plus productifs, même lorsque leur satisfaction initiale du système était grande.

Caractéristiques de changement

Ecklud et al. [Eck 96] ont identifié trois caractéristiques clés et nécessaires pour comprendre et analyser efficacement chaque changement considéré.

- L'***objectif*** d'un changement concerne l'ensemble des responsabilités du système que le changement affecte directement.
- La ***portée*** d'un changement par rapport à un modèle donné, fait référence à l'omniprésence des ramifications d'un changement à travers des artefacts élaborés dans ce modèle. Le terme «modèle» est utilisé pour désigner le produit final de toute phase du cycle de développement (ex. modèle d'exigences, modèle d'analyse, etc.).

- Le ***degré*** de définition d'un changement concerne la mesure avec laquelle les détails du changement potentiel sont connus.

La description du changement comprend la description de la modification intentionnelle et un ensemble de cas de changement associés, chacun décrivant un cas d'utilisation potentiel, ou un cas d'utilisation révisé. Chaque cas de changement, à son tour, a une relation avec les cas d'utilisation affectés par ce cas de changement (Figure 3.18).

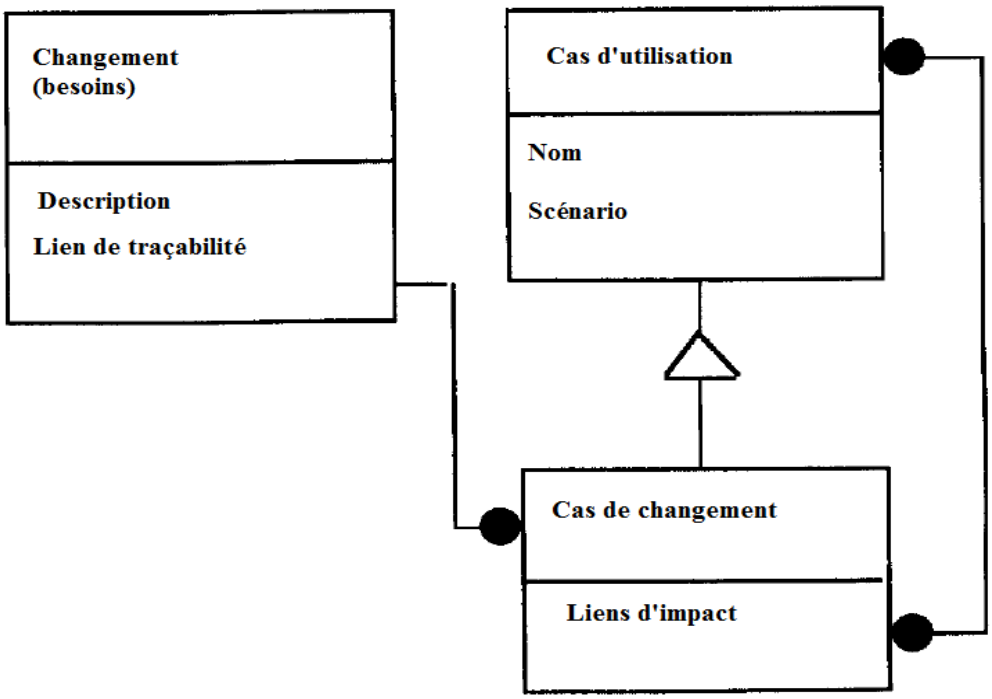


Figure 3.18 Changement, cas de changement et cas d'utilisation

La forme, existante actuellement, des cas de changement regroupe uniquement trois rubriques: un nom significatif, une probabilité d'occurrence et une description de l'impact que le changement peut avoir sur le système.

Les Cas de changement et la traçabilité

L'utilisation des cas de changement de l'analyse et l'adaptation au changement n'est pas en soi une méthodologie. Au contraire, elle constitue une technique qui peut être appliquée dans le cadre d'une méthode de développement orienté-objet, à condition que la méthode prenne en charge:

(a) l'inclusion des cas d'utilisation.

(b) l'inclusion et la maintenance des liens entre les différentes phases de la méthodologie.

L'utilisation efficace du modèle des cas de changement nécessite que la méthodologie prenne en charge la traçabilité d'un niveau de conception à un autre.

Autrement dit, la traçabilité fait référence à l'enregistrement explicite, sur papier ou dans un outil AGL, de liens entre artefacts d'un niveau à un autre dans la méthodologie.

Par exemple:

- Chaque objet d'analyse dans le modèle d'analyse est lié au cas d'utilisation qui l'a motivé (i.e. qui a provoqué sa construction).
- Chaque construction dans le modèle de conception est liée à l'objet d'analyse du modèle d'analyse à partir duquel il a été dérivé.
- Chaque test dans la suite de tests est lié au cas d'utilisation ou autre exigence qu'il est destiné à tester.

La figure 3.19 illustre cet exemple.

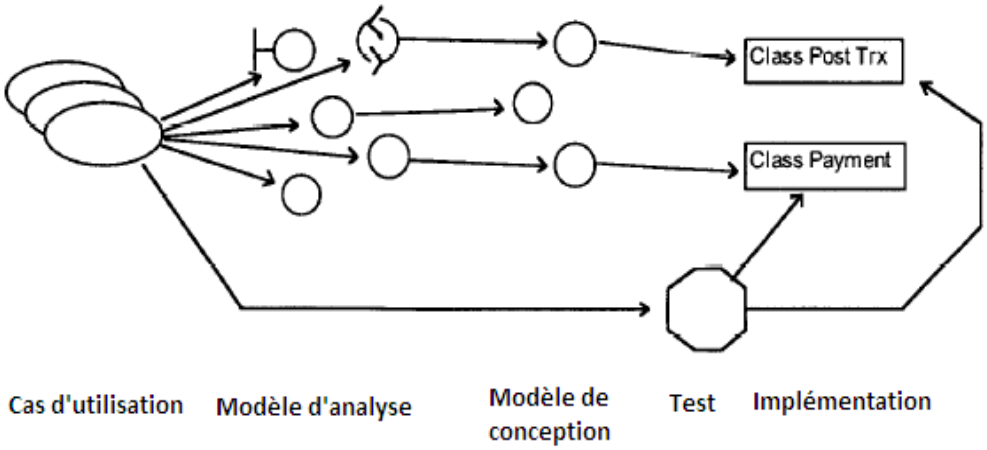


Figure 3.19 La traçabilité dans le développement de logiciels orienté-objet

Conclusion

L'importance de l'ingénierie des besoins dans le développement de systèmes logiciels a été longtemps établie et identifiée par des chercheurs et des praticiens. Cette partie de l'ingénierie du système se charge des activités de la découverte, de l'extraction, de la négociation, de la validation, de l'opérationnalisation et de la spécification des exigences. Plusieurs techniques et approches pour l'ingénierie des exigences ont été proposées dans la littérature, nous nous intéressons aux approches GORE et particulièrement le modèle orienté-but i*. La discipline RE du processus de développement RUP a été également présentée en détaillant chacune de ses activités. Enfin, nous avons abordé les concepts et modèles existants qui sont la base du processus d'ingénierie des changements/évolutions anticipés proposée dans la démarche ONTO-RUP tels que la méthode Futures wheel, et le formalisme de cas de changement.

CHAPITRE 4

ONTO-RUP : UNE DÉMARCHE DE DÉVELOPPEMENT DES SYSTÈMES ONTOGÉNÉTIQUES

Bien que la méthodologie RUP (Rational Unified Process) ne soit pas adaptée à la production des systèmes ontogénétiques, elle présente de nombreux atouts pratiques et jouit d'une notoriété mondiale. De ce fait, son adaptation pour les systèmes ontogénétiques est une voie prometteuse. Nous proposons ONTO-RUP, une démarche pour le développement des systèmes ontogénétiques, basée sur une extension du RUP. Pour cela, l'approche proposée prévoit un nouveau cycle de vie pour le RUP, en proposant l'adjonction d'autres disciplines, des éléments de modélisation et des directives pour mener la modélisation de l'ontogenèse. Pour cela, l'approche préserve tous les outils et atouts du RUP.

4.1 Approches Existantes et Ontogenèse

Plusieurs travaux de recherche ont montré que plus de 80% des coûts du cycle de vie du logiciel sont engagés après la livraison [Pim 11], [Sal 04], [Aur 05]. Par conséquent, il existe des raisons économiques importantes pour obtenir une meilleure compréhension du **pourquoi** et du **comment** les systèmes évoluent. La vision **comment** porte sur l'évolution du système logiciel comme une discipline d'ingénierie. Elle étudie les aspects plus pragmatiques qui aident le développeur du logiciel ou le chef de projet dans leurs tâches au jour le jour. Cependant, l'évolution des logiciels étudie également le processus de changement lui-même, l'analyse des débris du logiciel pour extraire les tendances, faire des

prédictions ou comprendre la nature même du phénomène de l'évolution du logiciel même (c'est à dire, il explore le **quoi** et **pourquoi**) [Mens 08].

4.1.1 Travaux Existants et Modélisation de l'Evolution

Les méthodes et les mécanismes standards actuels couvrent seulement une partie du processus de logiciel, et souvent ne supportent pas la phase de maintenance et d'évolution. La standardisation est rarement appliquée et les changements sont fréquemment implémentés manuellement. Ils n'existent pas actuellement des démarches de développement appropriées qui redonnent à l'évolution le statut de concept central.

La communauté du génie logiciel admet que les enjeux des nouvelles technologies se situent plutôt dans le contexte des méthodes de développement [Ben 12, Joh 05, Lim 11, Cha 01, Mes 08]. Ces dernières doivent faire face aux problèmes de l'évolution des systèmes logiciels mais aussi à leur complexité qui s'accroît à un rythme sans rapport avec nos capacités actuelles à les appréhender.

Sur le plan des démarches, il n'en existe pas encore, à notre connaissance, qui soient dédiées à l'approche ontogénétique qui mettent l'accent sur l'évolution et font que l'évolution devienne la préoccupation primaire de l'implémentation d'un système logiciel. La vision ontogénétique est radicale et sépare les changements dans deux catégories: Les changements anticipés et changements non anticipés.

Nous citons en premier les travaux de Pimentel and al. [Pim 11] qui représente notre source d'inspiration pour l'élicitation des évolutions anticipées des besoins. Les auteurs proposent une approche pour effectuer des changements sur les exigences exprimées par les modèles de but en se basant sur une représentation du futur fournie par la méthode « futures wheel ». Cependant, le travail se concentre principalement sur la façon dont les méthodes prospectives peuvent être utilisées pour l'élicitation des besoins futurs, et discute les impacts d'étudier le futur sur l'activité d'ingénierie des besoins. Les auteurs résument 17 méthodes prospectives où certaines de ces méthodes sont encore utilisées pour l'élicitation des besoins, mais pas pour étudier le futur; comme, par exemple, le cas des méthodes des scénarios et les méthodes participatives.

Une approche qui s'intéresse à la problématique de l'évolution sous l`angle des besoins est proposée dans [Eti 04] et [Sal 04] pour l'adaptation des systèmes d'information. Dans cette approche le processus d'évolution des besoins est modélisé en utilisant un méta-modèle et une typologie générique d'opérateurs exprimant les différents types des évolutions et ceci en utilisant le concept de l'écart (Gap en anglais).

D’autre travaux ont été proposés notamment, pour les systèmes d’information et les systèmes logiciels aux données-intensives où il est essential d’avoir une description claire et précise de schémas de bases de données. Hainaut and al. explorent en détail comment faire évoluer et migrer un schéma de base de données [Hai 08].

Nous pouvons citer pour l’évolution des systèmes d’information le travail basé sur de Hainaut et al. [Est 12]. Les auteurs proposent un Framework qui aide à identifier et contrôler les effets provoqués par l’évolution d’un système d’information et leurs conséquences.

Finalement, l’ingénierie de ligne de produits est une approche largement utilisée pour le développement efficace des systèmes [Bac 05]. Pour tenir compte des différences entre les produits logiciels, certaines adaptations des actifs de base sont habituellement exigées. Avec la sélection et la modification, un atout essentiel est choisi dans la base d'actifs et modifié ou configuré d'une façon planifiée. L'actif de produit résultant est encore reconnaissable comme une variante de l'actif de base d'origine.

Les changements non anticipés sont des évolutions qui ne se produisent qu’après la livraison du produit et qui ne sont pas prévues durant le développement du système. Bien qu’il soit possible de raisonner sur le futur [Gle 72, Pim 11, Lim 11, Kav 06, Gle 09, Eck 96, Zow 96, Fis 98], il n’est pas possible de prévoir toutes les possibilités de changement. Il est bien clair qu’il n’existe aucun modèle qui puisse éliciter quelque chose qui est totalement inattendu ou inconnu. Seulement, nous pouvons parler dans ce contexte des paradigmes comme le développement continu, amélioration continu de systèmes logiciels, concevoir des produits à faire évoluer dans le futur, etc. [Cis 14, Joh 05, Kru 01].

Une autre facette très importante et particulière de l’évolution est la mise à jour dynamique [Dow 01, Hic 01]. Elle permet à plusieurs applications importantes de s’exécuter continuellement et sans interruption. C’est le cas des systèmes critiques, tels que la commutation téléphonique, les transactions financières, la réservation des places d’avion et le contrôle du trafic aérien ou ferroviaire.

La littérature sur la maintenance du logiciel est beaucoup moins abondante que celle sur le développement du logiciel. Jusqu’aux années 1990, le thème de la maintenance était peu abordé dans les cursus universitaires d’informatique et de génie logiciel, et c’était en travaillant dans les organisations elles-mêmes que les ingénieurs logiciels s’initiaient aux spécificités de la maintenance et y développaient une expertise spécifique. La prise de conscience de l'importance de la maintenance durant le développement de systèmes logiciels a augmenté au cours des dernières décennies. De plus en plus, le logiciel n'est plus développé à partir de rien, mais constitue une poursuite du développement de logiciels existants [Phi 00].

Actuellement, beaucoup des travaux ont été proposés pour améliorer le processus d’évolution et de maintenance des systèmes logiciels, tel que la retro-ingénierie [Chi 90], la

compréhension de programmes [O'B 03], [Ber 07] et la réingénierie [Chi 90, Arn 93, Som 01].

Malgré l'existence de plusieurs méthodes de développements de logiciels, Ils en n'existent pas actuellement qui peuvent produire des systèmes ontogénétiques, c'est-à-dire celles qui prennent en charge les changements conformément à l'approche ontogénétique.

On peut distinguer deux catégories de méthodes de développement qui existent actuellement. Les méthodes traditionnelles axées-plan qui se basent sur un ensemble d'activités séquentielles. La deuxième catégorie est celle dite agile [Mes 08].

Ni l'approche traditionnelle ni l'approche agile ne peuvent produire des systèmes ontogénétiques. Nous proposons dans les sections suivantes notre approche Onto-RUP, une démarche pour le développement des systèmes ontogénétique basée sur une extension du Rational Unified Process.

4.1.2 Approches de Développement Existantes et Ontogenèse

Les systèmes ontogénétiques évoluent tout en restant opérationnels et en réponse à des stimuli internes. Les changements/évolutions anticipés sont analysés et codifiés durant les phases de développement du système, sous forme de patchs, qui s'exécutent si certaines conditions sont vérifiées. Comme signalé précédemment, ils n'existent pas dans la littérature des méthodes de développement qui prennent en charge la conception et l'implémentation des évolutions anticipées qui se déclenche automatiquement pendant le fonctionnement du système.

Pour faire évoluer les systèmes ontogénétiques pendant leur exécution tout en préservant leur disponibilité, c'est-à-dire leur continuité de service, les évolutions/changements non anticipés sont pris en charge dés leur arrivée et le système ontogénétique reste opérationnel et mis à jour dynamiquement, cette caractéristique existe dans certains systèmes logiciels mais elle est liée beaucoup plus à la plateforme du codage du logiciel qu'à la méthode de son développement et production.

Toutes les formes d'évolution sont traitées de la même façon. Ce point est important car les systèmes ontogénétiques présentent l'avantage de considérer les deux formes de changements (anticipés ou non) de manière identique. Cette vision est absente dans toutes les méthodes de développement existantes qu'elles soient traditionnelles ou agiles. Le cycle de maintenance du système logiciel est totalement indépendant du cycle de son développement (Figure 4.1).

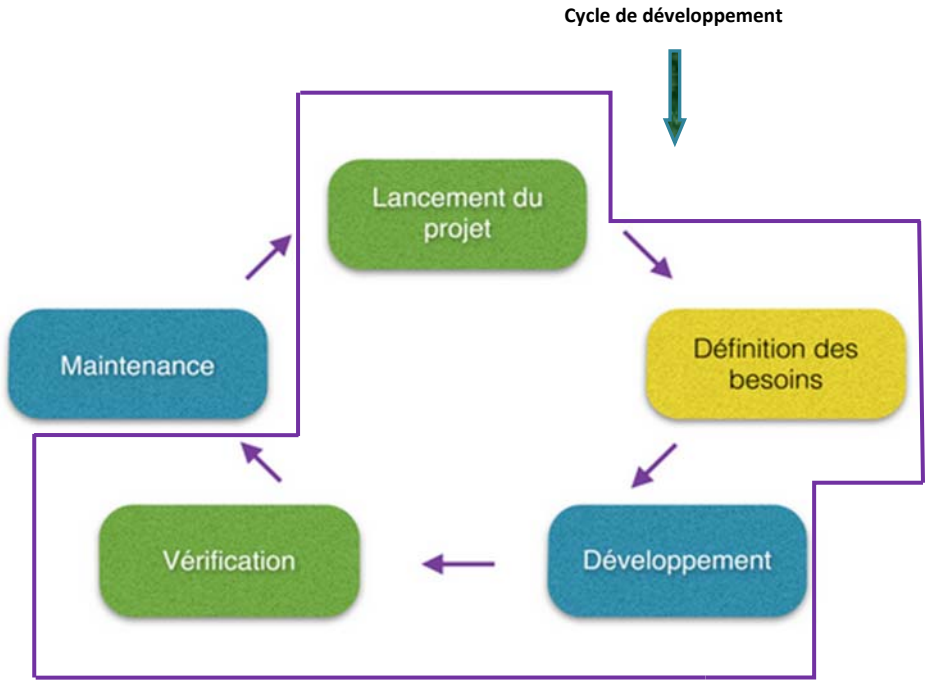


Figure 4.1 Cycle de développement et de maintenance de logiciel

Dans le développement des systèmes ontogénétique, nous proposons le paradigme de ***Développement Continu du Système Ontogénétique***.

4.2 Le Développement des Systèmes Ontogénétique : Aperçu

L'approche ontogénétique reporte la prise en compte des changements anticipés jusqu'à ce qu'ils deviennent nécessaires (i.e. certaines conditions se réalisent), suite à cela le code est altéré dynamiquement pour incorporer ces changements. Cette approche a l'avantage de maintenir le code performant, simple et reflétant les besoins réels actuels. Concrètement le code est débarrassé des structures alternatives au profit d'une évolution dynamique des comportements des objets en fonction des besoins réels actuels. Bien entendu cela suppose que le code du logiciel ait une structuration qui permet d'incorporer les changements de manière adéquate et que la plateforme d'accueil soit dotée de composants qui peuvent altérer le code du logiciel dynamiquement en fonction de la description du changement anticipé.

Les changements non anticipés sont des changements qui apparaissent une fois le logiciel livré. L'approche ontogénétique traite les deux types de changement de la même façon au niveau opérationnel. Cependant au niveau abstrait, le traitement diffère. La démarche

utilisée doit tenir compte d'un existant qu'il faut comprendre, dont il faut respecter certaines propriétés, etc.

Dans l'approche ontogénétique, le code du logiciel se compose de deux parties : Un code exécutable qui réalise les fonctionnalités du système, il est appelé Phénotype comme pour les organismes vivants, et un code nommé Génotype dont l'exécution façonne continuellement et dynamiquement le phénotype.

Autour du code ainsi formé, se trouve une suite de composants nécessaires représentant la plateforme d'exécution. Le schéma de la figure 4.2 décrit l'approche ontogénétique.

Les phases de développement du système ontogénétique concernent d'ingénierie et l'analyse aussi bien les besoins actuels des parties prenantes que ceux du futur anticipé. En effet, tous ses besoins seront des entées aux phases d'implémentation. Le Phénotype est opérationnel dès la livraison du produit, le génotype contient une structure spécifique contenant un ensemble de structures et contenant des versions anticipées du système au futur. L'arrivée d'un changement/évolution non anticipé représente une entrée au processus ***Analyse État actuel et futur du système***. Ce dernier effectue une analyse des changements, et les fournit à l'équipe de développement continu pour le codage, le test, et mise à jour du ***Composant Chargé de l'Altération du Génotype selon les Changements non Anticipés***.

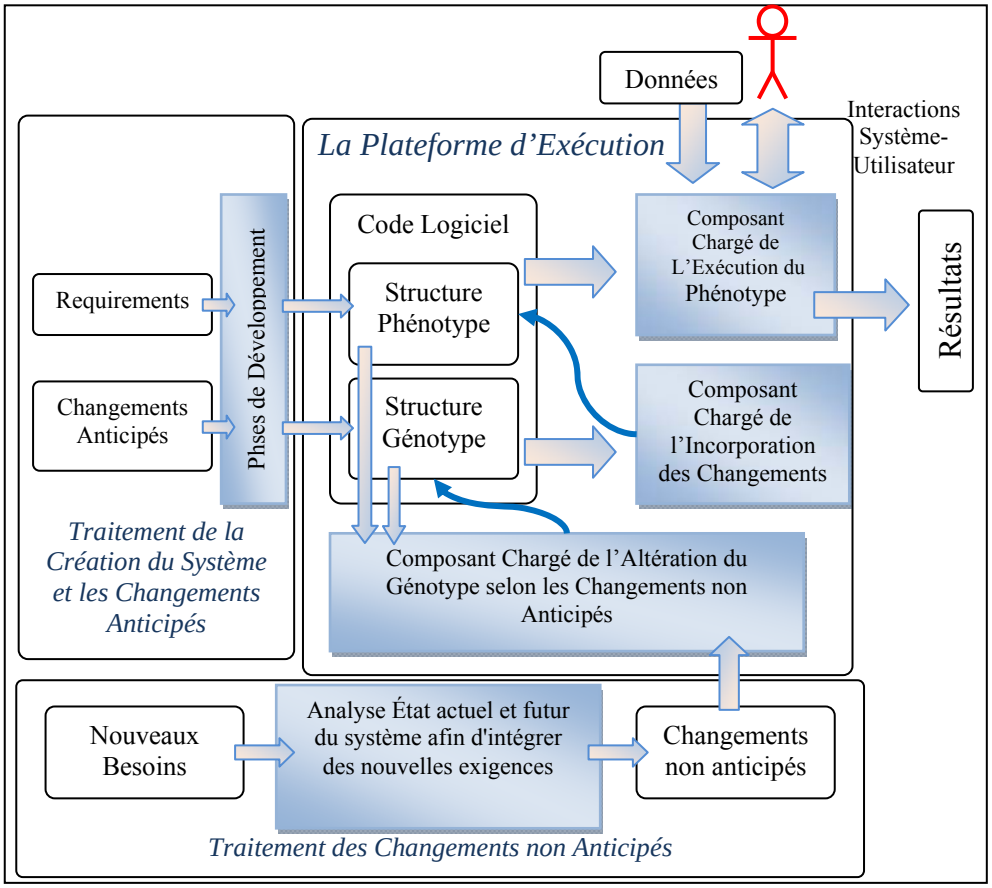


Figure 4.2 La vision ontogénétique d'un système logiciel

4.3 Onto-RUP : Une Extension du RUP

Le Rational Unified Process comme toutes les méthodes agiles a montré ses avantages et apports quant au développement des systèmes logiciels. C'est un cadre de processus fondé sur des pratiques avérées (voir Chapitre 2).

Le RUP et toutes les méthodologies de développement ne prennent pas en charge les évolutions futures des besoins durant le processus de développement. Nous proposons une extension du RUP pour l'adapter aux systèmes ontogénétiques. L'extension concerne l'adjonction de deux nouvelles disciplines et d'autres éléments de modélisation qui redonne à l'évolution un aspect fondamental dès le démarrage du processus du développement du système logiciel [Khe 14].

En raison des insuffisances des méthodes de développement classiques ou modernes pour conduire le développement des systèmes ontogénétiques, nous proposons l'extension du RUP. L'approche préserve tous les outils et atouts du processus RUP, et propose une extension pour l'adapter à notre contexte. L'extension porte sur deux nouvelles disciplines appelées respectivement Ant-Ev et Unant-EV et d'autres éléments de modélisation.

4.4 Onto-RUP : Un Nouveau Cycle de Vie pour le RUP

Nous proposons un nouveau cycle de vie pour le RUP. Le nouveau cycle concerne l'adjonction de nouvelles disciplines pour la modélisation des changements. Ceci va entrainer bien sûr une révision des objectifs des différentes phases du cycle de développement du RUP : Inception, Elaboration, Construction et Transition.

4.4.1 Axe Horizontal : Révision de la phase Inception

L'axe vertical de la structure représente l'aspect statique d'Onto-RUP. Donc quatre phases distinctes : Inception, Elaboration, construction et Transition. La phase d'inception du RUP comprend un ensemble d'activités riches et de conseils, ses objectifs sont :

- Comprendre le système à construire;
- Identifier les fonctionnalités clé du système;
- Déterminer au moins une solution possible;
- Comprendre les coûts, le calendrier et les risques associés au projet ;

- Décider sur le processus à suivre et les outils à utiliser.

La prise en compte le l'ingénierie des changements/évolution anticipés durant cette phase nous a conduit à l'adjonction des nouveaux objectifs suivants :

- Etude des tendances et évènements futurs
- Identifier les changements/évolutions anticipés;
- Comprendre les coûts, le calendrier et les risques des tendances futures

4.4.2 Axe Vertical : Nouvelles Disciplines

L'axe vertical de la structure d'Onto-RUP, représente l'aspect statique de la structure. Cet axe représente toutes les disciplines du RUP en plus les deux disciplines proposées dans cette approche et qui sont :

- la discipline AntRE et
- la discipline UnantRE.

4.5 La Discipline Ant-Ev d'Onto-RUP

La discipline ***Requirements*** du RUP utilise le modèle des cas d'utilisation comme des artéfacts englobant les besoins fonctionnels du système logiciel à produire. Le RUP est un cadre de processus extensible [Kro 03]. Nous proposons une discipline additionnelle Ant-RE, représentant une extension des disciplines RUP pour mener l'anticipation des besoins de changement/évolution.

Au début nous nous somme intéressés à la réponse de la question qui s'imposait fortement: « Existe-t-il un modèle qui représente une prospection du futur des cas d'utilisation? », dont la réponse était « Oui, il existe un modèle dit le ***modèle des cas de changement***, proposé par E. Ecklud [Eck 96] et qui représente une altération du modèle des cas de d'utilisation. Cependant, nous avons proposé dans [Khe 09a, Khe 09b] une démarche d'extraction de changement/évolution anticipé à partir du modèle des cas d'utilisation en utilisant une combinaison d'approches classiques d'élicitation des besoins tels que : Brainstorming, Interview, Questionnaires, et Groupwork. Néanmoins, ces approches ne sont pas appropriées pour l'étude des changements/évolutions des évènements et tendances futures du système logiciel à produire. Le processus d'élicitation des exigences de changement/évolution est un processus qui doit se baser sur un raisonnement intentionnel, à commencer par des objectifs de haut niveau et d'affiner ces objectifs jusqu'à atteindre ceux qui sont opérationnalisable. L'approche GORE se base sur la définition des exigences comme étant des buts qui peuvent être divisés et raffinés [Lam 09]. Cependant,

une possibilité de modéliser des évolutions/changements est l'utilisation des modèles orientés-but (cf. chapitre 3). La modélisation des besoins orientés buts a été proposée durant l'élicitation des besoins pour décrire le comportement organisationnel actuel. Également les techniques d'analyse orientées buts ont été utilisées dans le contexte de la négociation des besoins pour aider le raisonnement sur la nécessité de changement organisationnel, et de fournir le contexte dans lequel se produit la délibération durant l'ingénierie des besoins [Kav 06, Pim 11, Nur 02]. En effet, i* [Yu 95, Yu 97] fournit un mécanisme convenable pour représenter les comportements alternatives d'un système à travers les liens « moyen-finalité». Cette caractéristique intègre facilement les besoins futurs avec ceux actuels.

Cependant, nous proposons l'élaboration d'un modèle orienté but au cours des activités de la discipline proposée. L'altération/adaptation du modèle orienté but du système sert à représenter une altération du modèle des cas d'utilisation, d'où la création du modèle des cas de changement.

4.5.1 Objectifs de la discipline Ant-Ev

Avant de citer les objectifs de la discipline d'ingénierie des besoins de changement, nous rappelons ceux de la discipline RE du RUP.

L'élicitation des besoins pour un système logiciel est une activité conceptuelle complexe, nécessitant des efforts importants de mise en œuvre. C'est est un processus d'une grande importance permettant de mieux spécifier le système logiciel à produire. Cette activité comprend les étapes suivantes :

1. Comprendre le domaine de l'application
2. Identifier les sources des exigences
3. Analyser les besoins des parties prenantes
4. Choix des techniques, approches et outils les plus appropriés.

Les objectifs de la discipline de la gestion des exigences du RUP se résument donc dans les points suivants :

- Établir un accord entre les intervenants sur les objectifs du système à développer.
- Fournir aux développeurs du système une meilleure compréhension des exigences logicielles.
- Définir les limites du système à développer.
- Fournir un plan initial des itérations à réaliser.

- Fournir des estimations initiales des coûts, des échéanciers pour développer le système.
- Définir les interfaces graphiques en se basant sur les besoins des utilisateurs.

Le résultat de la combinaison de ces étapes est la spécification des besoins actuels du système en cours de développement.

Nous ajoutons aux objectifs cités précédemment les objectifs spécifiques à la discipline Ant-Re :

- ⇨ La construction du modèle orienté but associé à l'analyse des Extraction des évènements et tendances futures
- ⇨ La construction du modèle orienté but altéré ou adapté.
- ⇨ L'analyse du mapping i*/cas utilisation
- ⇨ La construction du modèle des cas de changement

4.5.2 Activités de la Discipline Ant-RE

La discipline Ant-RE propose, d'autres activités pour l'anticipation des évolutions futures. Ces activités permettent d'étudier les possibilités de prévoir les besoins futurs du système.

L'ingénierie des besoins doit poser la question POURQUOI développer un système et aider les parties prenantes du projet à y répondre [Rol 03]. Le rôle de l'ingénierie des besoins est de déterminer ensuite les fonctionnalités que le système doit mettre en œuvre pour aider à la satisfaction de ces buts et identifier les contraintes qui restreignent la mise en œuvre de ces fonctions. Ces buts, fonctions et contraintes, constituent les 'besoins' qui doivent ultérieurement être convertis en une spécification précise permettant le développement du système. Dans ce sens nous utiliserons le modèle des cas d'utilisation de la discipline RE du RUP comme entrée aux différentes activités de la discipline proposée. La discipline Ant-RE propose, d'autres activités pour l'anticipation des évolutions futures. Ces activités permettent d'étudier les possibilités de prévoir les besoins futurs du système.

Supposons qu'un modèle orienté-but et son modèle des cas d'utilisation correspondant sont déjà établit lors du processus d'analyse et spécification des besoins. Ces deux modèles représentent des entrées aux activités de la discipline proposée. Le développement du système logiciel se produit dans un contexte où les processus organisationnels sont bien établis [Bub 94]. La modélisation des besoins orientée but a été proposée durant l'élicitation des besoins pour décrire le comportement organisationnel actuel du système [Kav 06, Lam 01]. Dans ce contexte, nous utilisons le modèle i*, décrit en 3.2.2, pour exprimer l'ensemble des besoins actuels du système ontogénétique. Le choix d'utilisation

des modèle orienté-but et son adaptation a été inspiré des travaux de J. Pimentel et al. [Pim 11].

Également les techniques d'analyse orientées but ont été utilisées dans le contexte de la négociation des besoins pour aider le raisonnement sur la nécessité de changement organisationnel, et de fournir le contexte dans lequel se produit la délibération durant (Figure 4.3).

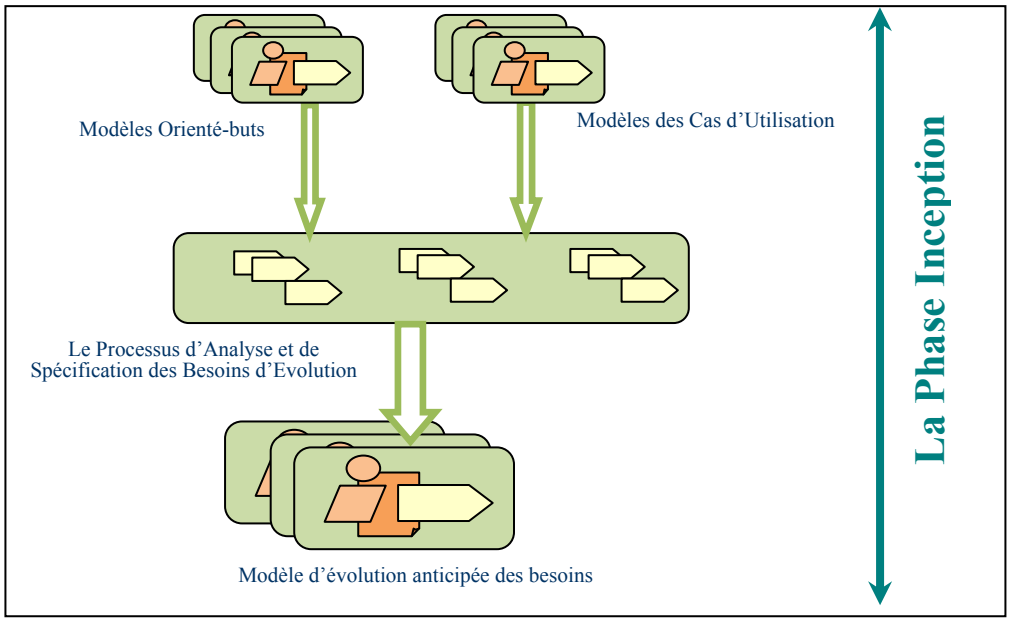


Figure 4.3 Les entées du processus d'analyse de l'évolution anticipée des besoins

4.5.2.1 Activité : Construction du modèle des buts

L'ingénierie des besoins doit d'abord s'intéresser aux buts du contexte organisationnel du système à développer parce qu'ils permettent de comprendre les raisons justifiant son développement. Le but de l'élaboration du modèle des buts est de permettre le raisonnement sur la nécessité de changement ultérieurement, c'est-à-dire juste après l'extraction des tendances et évènements futurs. L'intention de Changer est un but qui doit être satisfait avant d'être convertis ultérieurement en une exigence de changement anticipé dans le système à produire. Le modèle des buts est conforme à la notation i* [Yu 95, Yu 97] décrite en 3.2.2.

Exemple

Le but principal du « Movies For Me » est d'informer les personnes intéressées sur les horaires des téléfilms, qui réalisé par les tâches "Découvrir les horaires des téléfilms" et "Afficher les horaires téléfilms". La tâche "Découvrir les horaires des téléfilms" est achevée avec les deux tâches "Découvrir films Chaîne A" et "Découvrir films chaîne B". Chacune

d'elle est décomposée respectivement, en " Analyse site web de la chaîne", "Obtenir description films", "Obtenir date et horaires films" et "Obtenir images films".

Le but "Afficher calendrier téléfilms" est achevé avec les tâches "Liste des films sur site web" et les films peuvent être regroupés par jour d'exposition ou par chaîne, Figure 4.4.

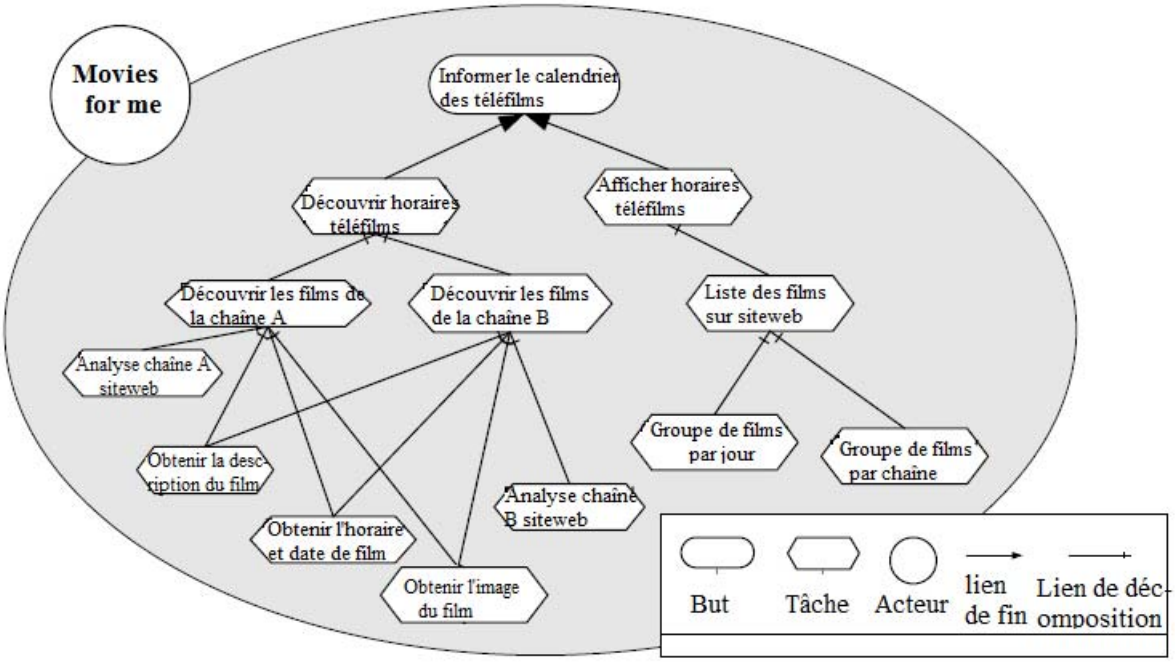


Figure 4.4 Le Modèle orienté-but du système « Movies for Me » [Pim 11]

4.5.2.2 Activité : Élicitation des Besoins Anticipés: Analyse Futures wheel

La méthode Futures wheel [Gle 72] semble plus appropriée pour l'élicitation des besoins futurs, car elle offre un processus qui :

1. fournit une image claire des événements futurs qui pourraient avoir une incidence sur le système,
2. est facile à comprendre et à utiliser par les parties prenantes,
3. nécessite moins d'effort que les autres approches [Gor 04], et est plus approprié pour l'ingénierie des besoins [Pim 11] et par conséquent, évite de compromettre le calendrier du projet.

Cette dernière caractéristique est très importante pour préserver l'un des principes du processus RUP : "la réduction de la complexité" du système à construire.

Comme décrit en 3.4.1, le processus Futures wheel comprend deux étapes, l'élicitation des évènements et leurs conséquences. A la fin du processus, on aura autant de diagrammes

que des évènements futurs. Un modèle d'un seul évènement se représente sous la forme présentée en figure 4.5.

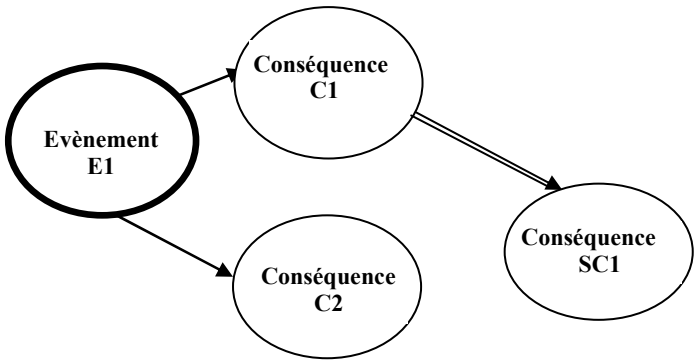


Figure 4.5 Exemple de notation Futures wheel

On attribue pour chaque évènement, un modèle Futures wheel. Un graphique dans lequel on peut voir toutes les conséquences possibles sortantes du cercle de l'évènement. L'événement est représenté par un cercle avec une bordure épaisse. Les conséquences sont représentées par un cercle avec une bordure normale. L'événement principal est lié aux conséquences primaires par une flèche sur une seule ligne; les principales conséquences sont liées aux conséquences secondaires par une ligne doublée et fléchée, et ainsi de suite.

Dans la figure 4.6 l'évènement Adoption réussie de la télévision numérique terrestre a comme conséquences directes :

- Plus de chaînes TV disponibles
- La disponibilité des chaînes TV dans les appareils mobiles
- La diffusion EPG (Description électronique du contenu diffusé)
- Et la haute définition

Par exemple une sous-conséquence de la conséquence « Plus de chaînes TV disponibles » sera : Le système aura besoin de rassembler les données sur des chaînes qui n'existent pas encore.

A son tour « Haute Définition » a une autre conséquence : Les gens sont plus susceptibles de regarder des films à la télévision.

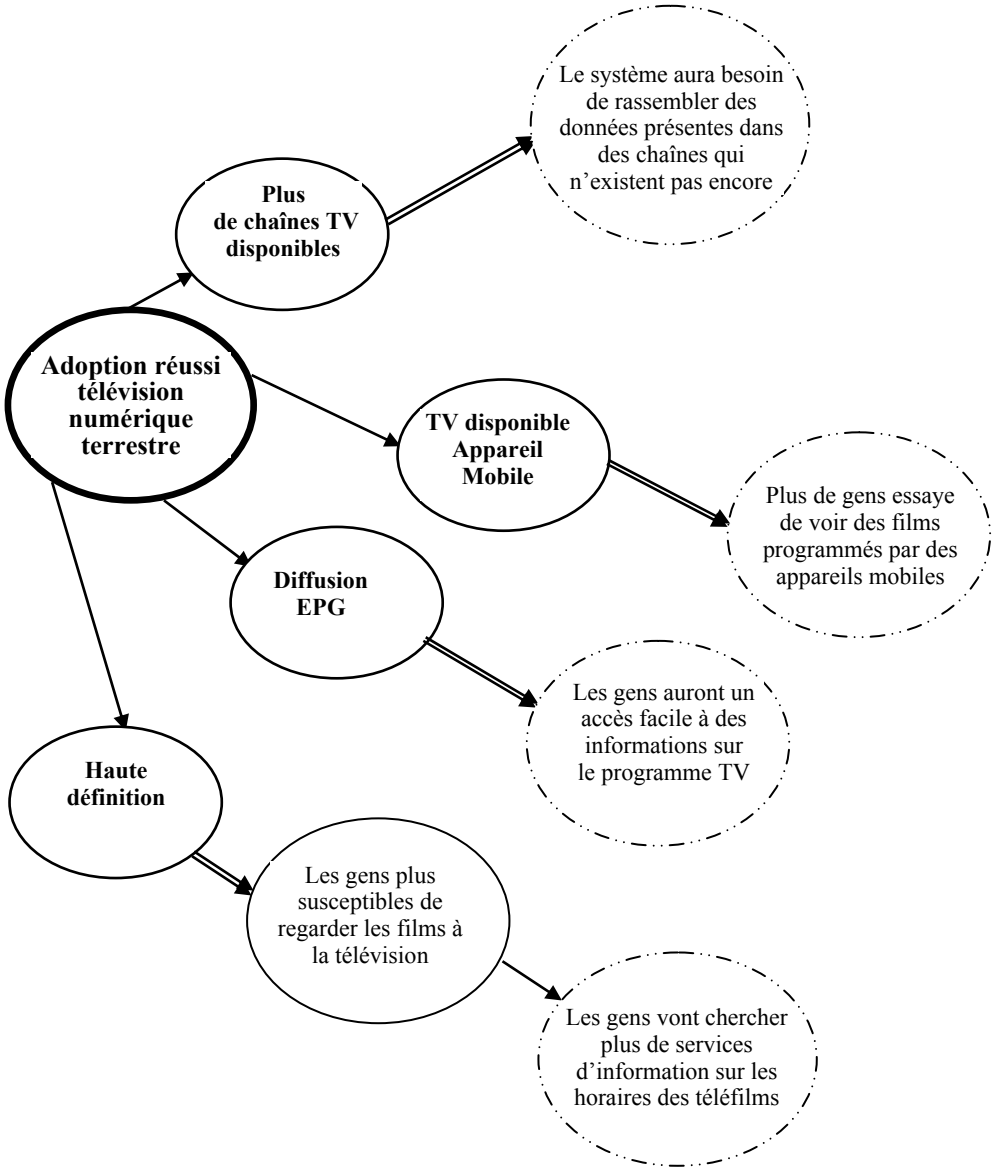


Figure 4.6 Le modèle Futures wheel pour l'évènement ″ adoption réussie de la TV terrestre numérique″

Un autre évènement analysé est « Croissance économique ». Comme montré dans figure 4.7, les conséquences considérées et relatives au domaine sont :

- Augmentation de demande de TV payante et,
- Augmentation d'accès à l'internet.

« Plus de personnes accédants à Internet » est une conséquence secondaire de cet évènement. Encore une fois, pour chaque conséquence feuille, des conséquences directes sont identifiées.

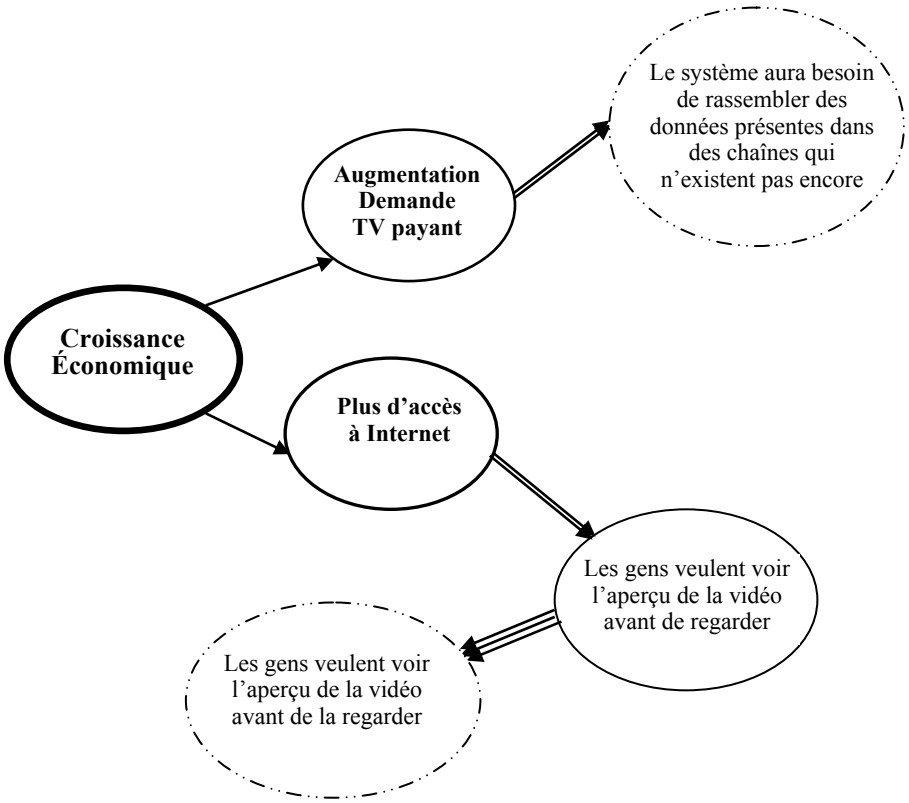


Figure 4.7 Le Modèle futures wheel de l'évènement "croissance économique" [Pim 11]

4.5.2.3 Activité : Adaptation du Modèle des Buts

Le résultat du processus d'élicitation des besoins futurs forme l'ensemble des Futures wheel, en d'autres termes le résultat de ce processus est en ensemble d'évènements associés à leurs conséquences. La modélisation des évolutions anticipée est un processus qui utilise le modèle des Futures wheel comme entrée. Cette modélisation s'effectue en deux étapes.

Étape 1 : Extension du Modèle Futures wheel

Le résultat du processus d'élicitation des évolutions anticipées est la construction du modèle Futures wheel qui contient l'ensemble des évènements ainsi que leurs conséquences associées. À ce stade, il y a encore un grand écart entre les conséquences et la configuration système requise. Ainsi, pour chaque conséquence feuille, c'est-à-dire les conséquences qui n'ont pas d'autres conséquences, on se demande comment cette conséquence affecte-t-elle le système ? On qualifie ces conséquences de directes, car elles sont directement liées au système. Pour expliciter quelles sont les conséquences directes, il est préférable de les représenter comme des cercles avec une bordure en pointillée (Figure 4.8).

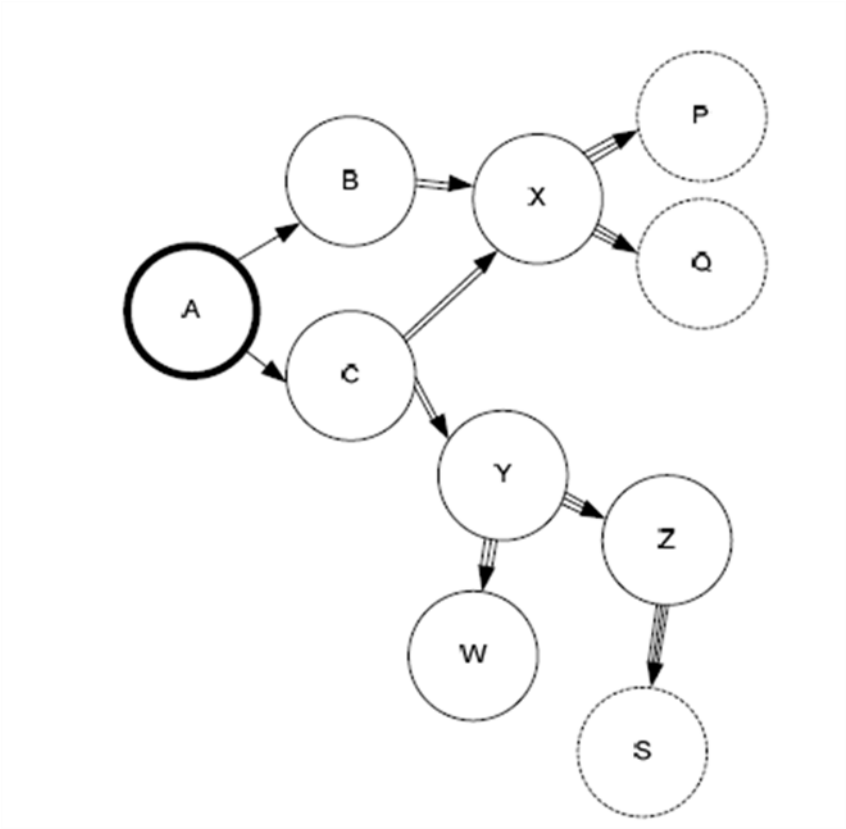


Figure 4.8 Exemple de notation du modèle Futures wheel étendu

Dans l'exemple étudié, les deux figures (Figure 4.6) et (Figure 4.7) représentent respectivement les modèles Futures wheel étendu des deux évènements adoption réussie de la TV terrestre numérique et "croissance économique".

Étape 2 : Création du Modèle Orienté-But Adapté

La deuxième étape dans le processus proposé est l'altération du modèle orienté-but existant. Ce modèle ne remplace pas le modèle initial, mais utilise le modèle en entrée pour la création d'un nouveau modèle dit ***le modèle orienté-but altéré***. L'altération du modèle des besoins orienté-but s'effectue après l'analyse du modèle Futures wheel. Cette analyse permet de voir, vérifier, et comparer pour répondre à la question « comment ces conséquences pourront-elles altérer le modèle orienté-but existant? ».

Les analystes doivent par la suite répondre à cette question en effectuant les changements appropriés dans le modèle initial conformément aux conséquences directes collectées lors de l'analyse Futures wheel.

Le tableau 4.2 montre les changements qui ont été apportés au modèle des buts, pour chaque conséquence directe. Pour résoudre les conséquences A et F, la tâche "Ajouter

ajout un support pour une nouvelle chaîne " doit être ajouté. De cette manière, le système devra être capable d'obtenir des informations à partir d'une nouvelle chaîne informé par son utilisateur. Le softgoal "Soyez moteur de recherche convivial" a été ajouté pour aborder la conséquence B, de sorte que lorsque les gens recherchent le calendrier des films TV, ils atteignent le site web "Movies For Me ". Il était également ajouté un softgoal et une tâche qui contribue positivement à ce softgoal.

L'intention d'aborder la conséquence C est le softgoal "Portabilité". En outre, afin de satisfaire le softgoal "Portabilité", la tâche "Liste des films sur un site web" est décomposée en trois tâches supplémentaires, de telle sorte que chaque type d'appareil dispose d'un site web spécifique. La conséquence E est abordée par l'ajout de la tâche "Obtenir Teaser/Trailer", de sorte que le système peut fournir un aperçu de la vidéo des films pour ses utilisateurs.

Conséquences Directes	Impact Spécifique
(A) Le système aura besoin de recueillir des données à partir des chaînes qui n'existent pas	Ajouter la tâche "ajout un support pour une nouvelle chaîne "
(B) Les gens vont chercher plus de services qui leur informent sur le calendrier des chaînes TV.	Ajouter le softgoal " Etre moteur de recherche convivial"; ajout le softgoal "Utiliser de bon mots clé"; ajouter la tâche "Utiliser liens sponsorisés".
(C) Plus les gens vont essayer de voir le calendrier des films à travers des dispositifs mobiles.	Ajout le softgoal "Portabilité" ; Ajout la tâche "site web pour PC et PC portable" ; Ajout la tâche " site web Spécifique pour des dispositifs mobiles" ; Ajout la tâche "Site web Spécifique pour TV".
(D) Les gens auront un accès facile aux informations sur le calendrier de la télé	Aucun
(E) Les gens voudront voir un aperçu vidéo du film avant de le regarder réellement.	Ajouter la tâche "ajout un support pour une nouvelle chaîne "
(F) Le système devra recueillir des données à partir de chaînes de télévisions payantes.	

Table 4.2 Information de traçabilité des conséquences directes et leur impact sur le modèle des buts

Toutes les conséquences directes résultats de l'analyse ou la modélisation Futures wheel sont prises en compte pour la création du modèle orienté-but adapté.

Le modèle orienté-but altéré du système « Movies for Me » est donné dans la figure 4.10.

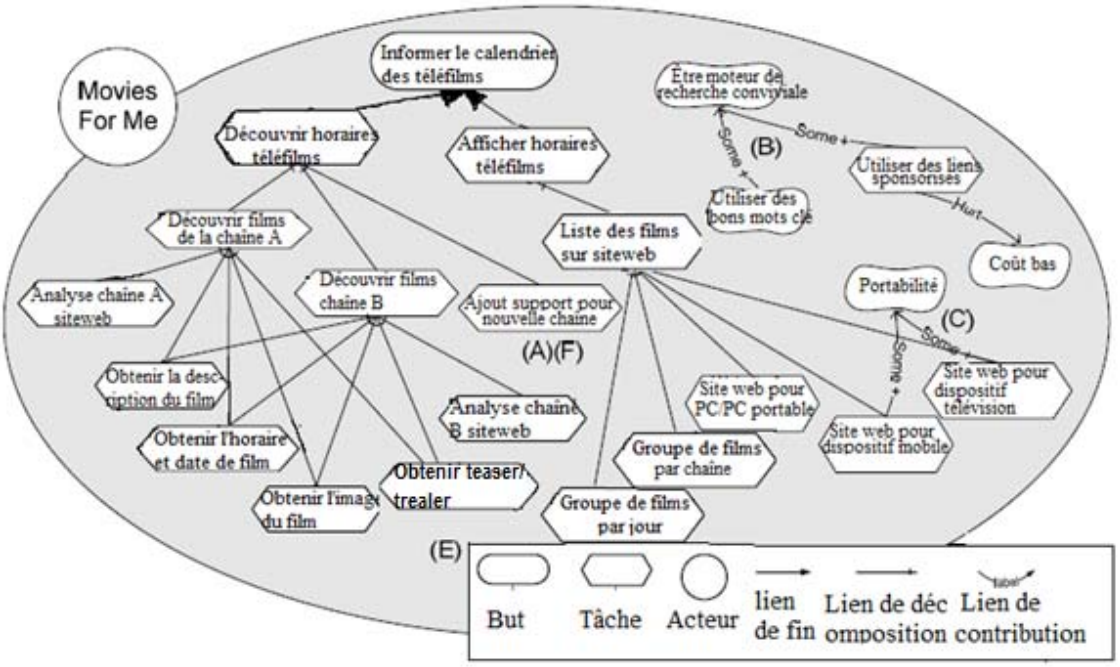


Figure 4.10 Le Modèle orienté-but altéré du système « Movies for Me » [Pim 11]

4.5.2.4 La Construction du Modèle des Cas de Changement

La discipline RE du RUP utilise le modèle de cas d'utilisation comme étant la structuration et la spécification des besoins fonctionnels du produit en cours de production. Donc, notre approche préserve l'utilisation des cas d'utilisation comme étant le noyau de la discipline d'ingénierie des exigences. A ce stade, une question se pose : quelle sera la relation entre le modèle des buts, le modèle des buts adapté et le modèle des cas d'utilisation ?

Nous avons proposé dans [Khe 9a], [Khe 9b] une approche basée sur les cas de changements [Eck 96] pour la modélisation des changements à partir du modèle des cas d'utilisation et une extraction d'un ensemble d'opérateurs appelés opérateurs de changement. Néanmoins, comme signalé précédemment, il est plus adéquat pour le raisonnement sur la nécessité de changement d'utiliser l'approche GORE. Wautelet et al. ont proposé dans [Wau 13a, Wau 13b] un mapping pour intégrer le modèle i* dans le RUP/UML.

Donc, il est possible d'extraire un modèle de cas d'utilisation RUP/UML à partir d'un modèle orienté-but i*.

Le modèle de cas de changement constitue le différentiel qui existe entre le modèle de cas d'utilisation mappé à partir du modèle des buts initial et le modèle de cas d'utilisation mappé à partir du modèle des buts adaptés suite à l'évolution.

Un Mappage du modèle i*/cas d'utilisation

Pour Cockburn un cas d'utilisation (use case) doit être attaché à un but (goal) déployé sous la forme d'une collection de scénarios. Chaque action d'un scénario peut à son tour, être vue comme un but associé à un nouveau cas d'utilisation de la collection des cas d'utilisation en cours de construction. Il y a donc une forme de récursivité illustrée par les liens (Figure 4.11).

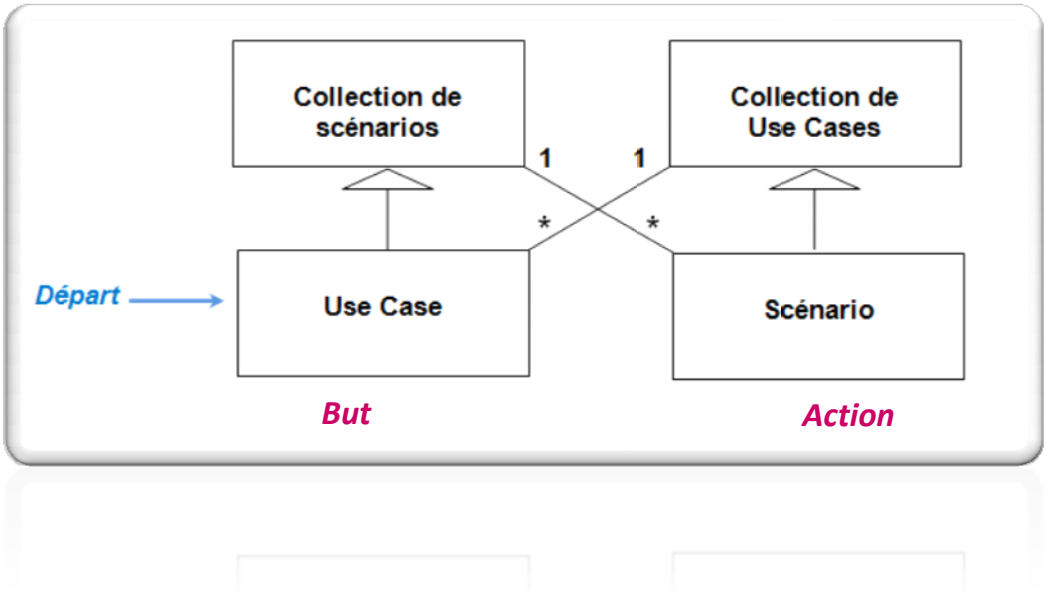


Figure 4.11 Le lien récursif entre Cas d'utilisation et scénario de Cockburn

L'intégration d'un modèle orienté but dans le modèle de cas d'utilisation existe déjà [San 02, Lee 99].

Des travaux ont étés menés [Wau 13a, Wau 13b] pour intégrer le modèle i* dans le RUP/UML, en particulier le modèle des cas d'utilisation métier du RUP. Donc le passage d'un modèle orienté-buts tel que le modèle adopté dans notre approche i* vers le modèle des cas d'utilisation du RUP est possible.

La table 4.1 résume les résultats de recherches [Wau 13a], [Wau 13b]. La notation graphique est documentée dans [Wau 13a].

Élément i*	Méthode
But Tâche Resource	Cas d'utilisation métier Réalisation cas d'utilisation métier Entité métier But métier
Acteur Position Agent Rôle Limite acteur	Acteur métier Contrôle Acteur Agent métier Package
(Stratégique) Dépendance Association acteur	Dépendances ou Instance Généralisation
Extrémité Décomposition Contribution	Inclure Agrégation Association unidirectionnelle

Table 4.2 Le modèle de mapping i* [Wau 13b]

⇨ ***But dur (hard) :*** Dans une dépendance de but, le *depender* dépend du *dependee* pour provoquer un certain état métier dans le monde.

Elément choisi : *Cas d'utilisation métier.*

Selon la base de connaissances du RUP, un cas d'utilisation métier (classe) définit un ensemble d'instances de cas d'utilisation métier dans laquelle chaque instance est une séquence d'actions que l'organisation effectue pour donner à un acteur métier particulier un résultat d'une valeur observable. L'élément de cas d'utilisation métier (BUC) élément a été choisi parce qu'il est situé au niveau métier (i.e. organisationnel) comme le but i* qui donne un résultat d'une valeur observable.

⇨ **Tâche :** Dans une dépendance de tâche, le *depender* dépend de la *dependee* pour mener à une activité. Les noms de *dependum* est une tâche qui spécifie la façon dans laquelle la tâche est effectuée, mais pas pourquoi. Le depender a déjà pris des décisions sur la façon dont la tâche doit être exécutée.

Elément choisi *: Réalisation de Cas d'utilisation*

Suite à la base de connaissances de RUP, une réalisation de cas d'utilisation métier décrit comment les employés d'entreprises, entités commerciales, et événements métier collaborent pour effectuer un cas d'utilisation particulier. Cela correspond à l'objectif de la tâche d'i* et est conforme au choix pour l'élément puisque le BUC (cas d'utilisation métier) a été sélectionné à ce stade.

⇨ **Softgoal:** Dans une dépendance softgoal, un depender dépend de la Dependee pour effectuer une tâche qui répond à un softgoal. Un softgoal est similaire à un but, sauf que les critères de réussite ne sont pas fortement définis à priori. Le sens

de Softgoal est élaboré en fonction des méthodes qui sont choisies pour poursuivre le but.

Elément choisi : But métier

Selon la base de connaissances de RUP, un objectif métier est une exigence qui doit être satisfaite par l'organisation. Les buts métier décrivent la valeur désirée d'une mesure particulière à un moment donné dans le temps et peut donc être utilisé pour planifier et gérer les activités de l'entreprise. Cette définition correspond mieux à l'objet du Softgoal

⇨ **Actor Boundary:** Elle indique les limites intentionnelles d'un acteur particulier.

Elément choisi: Package

Selon la base de connaissances de RUP, un mécanisme d'usage général pour organiser les éléments en groupes et le package. Les packages peuvent être imbriqués dans d'autres packages.

Les Cas de Changement : modèle pour les évolutions anticipées

Nous avons proposé l'extension de ce formalisme dans le but de décrire aussi bien les nouveaux besoins d'un système que les modifications des besoins existants. Les cas de changement se caractérisent par leur simplicité ainsi que leur puissance dans l'extraction des changements pouvant altérer un système. Les cas de changement forment dans leur ensemble le modèle des cas d'utilisation altérés.

Chaque cas de changement est lié à un cas d'utilisation qu'il affecte. Les cas de changement renferment les rubriques suivantes :

Un nom. Il permet d'identifier le cas de changement. Il a une forme très simple qui le met éventuellement en relation avec les cas d'utilisation

Un contexte d'occurrence. Il donne une idée sur le contexte dans lequel le cas de changement a lieu.

Cas d'utilisation affecté. Nomme le cas d'utilisation qui est touché par le changement et indique de quelle manière il l'est.

Raison du changement. Indique les motivations du changement décrit. Les raisons peuvent être technologiques, fonctionnelles, législatives, …

Probabilité de l'occurrence du changement. Indique si le changement est peu probable, probable, certain, ...

Nature du changement. Les changements peuvent être permanents ou éphémères, rares ou fréquents, à occurrence immédiate ou retardée, … Une combinaison de ces attributs est possible.

La portée du changement. Donne une idée sur les parties du système affectées par le changement.

Un facteur déclencheur. Le facteur qui déclenche le changement peut être un acteur (le système lui-même ou tout autre acteur en interaction avec le système tel que l'horloge) ou une condition qui se réalise.

Les autres parties sont identiques à celles des cas d'utilisation.

La table 4.2 donne un exemple de cas de changement. Il s'agit du cas de changement « Retrait sans solde pour client fidèle » . La banque dispose d'une multitude de distributeurs automatiques de billet de banque (DAB) installés dans ou à proximité de ses agences. Ces DAB permettent aux clients d'effectuer différentes transactions bancaires en libre-service. La banque souhaite devenir plus compétitive en fidélisant ses anciens clients. Une nouvelle règle de gestion est instaurée. Elle autorise les clients à débiter leur compte d'une somme qui excède le solde de leur compte de 200 unités. Cette évolution est déclenchée après deux années de la date de création du compte client. Le cas d'utilisation Retrait via le DAB reste opérationnel pour toutes les transactions de retrait concernant les comptes des clients non anciens, i.e. les clients des comptes datant de moins de deux années.

Par contre, pour les anciens clients, i.e. ceux dont leurs comptes datent d'au minimum 2 années, le retrait d'argent via le DAB se fait par un cas d'utilisation alternatif nommé « Retrait sans solde pour client fidèle ». Ce dernier représente un cas de changement pour le cas d'utilisation usuelle de retrait d'argent via le DAB.

Ce changement affecte un seul cas d'utilisation Retrait via le DAB. L'acteur principal étant le client qui désire d'effectuer un retrait d'argent via DAB. Le changement s'effectue exactement lors de la vérification du système du montant à retirer. Cependant, la conception et l'implémentation de ce changement peut être effectué en utilisant un patch.

Donc déterminer si le patch doit persister ou doit être incorporé dans le modèle objet (i.e. dans les méthodes) dépend du changement lui-même, mais aussi de certaines qualités de service (e.g. performance et conformité du modèle à la réalité).

La traçabilité dans ce cas de changement est un lien entre le cas de changement et l'ensemble des cas d'utilisation qu'il affecte. Dans ce cas de changement « Retrait sans solde client fidèle », le lien de traçabilité étant un lien logique reliant ce cas de changement avec le cas d'utilisation « Retrait via DAB » qu'il affecte. Tracer les cas de changement aux cas d'utilisation requiert fondamentalement une approche systémique replaçant chaque choix dans les phases d'analyse, de codage et de test.

Change case name	Retrait sans solde pour client fidèle
Occurrence context	La banque souhaite devenir plus compétitive en fidélisant ses anciens clients
Concerned use cases	Retrait via le DAB. Ce cas d'utilisation reste valable
Reasons behind the change	Nouvelle règle de gestion visant à fidéliser les clients
Likelihood	Le changement est certain et s'opérera pour les clients affiliés à la banque depuis deux années ou plus
Nature of the change	Permanent
Scope of the change	Touche seulement les clients affiliés depuis deux ans ou plus. Tous les cas d'utilisation utilisant Retrait via DAB, doivent utiliser le nouveau cas d'utilisation lorsqu'il s'agit de compte datant de deux ans et plus
Triggering factor	Deux ans après l'ouverture d'un compte
Context of use	La banque dispose d'une multitude de DAB installés dans ou à proximité de ses agences. Les clients peuvent lancer une opération de retrait qui leur permet d'obtenir des billets de banque
Main Actor	Le client
Preconditions	Le client souhaite effectuer un retrait et est authentifié
Postconditions	Le solde du client est inchangé ou diminué du montant retiré. Le solde est supérieur ou égal à -200
Used use cases	**UC1:** Arrêt d'interaction avec un client
Description	1: Le système invite le client à introduire un montant pour le retrait. Le client introduit le montant 2: Le système vérifie que le montant est inférieur ou égal à 'solde+200' et délivre les billets 3: Arrêt d'interaction avec UC1
Alternatives	2a: Si le montant est supérieur à 'solde+200', informer le client et aller à l'étape 1. Au bout de trois tentatives, aller à l'étape 3

Table 4.2 Cas de changement pour la fidélisation des clients

Les Cas de Changement : Une Altération des Cas d'Utilisation

Le modèle orienté-but altéré donne une possibilité de raisonner sur le modèle des cas d'utilisation existant pour créer le modèle des cas de changement.

Aussi, les diagrammes de cas d'utilisation permettent de définir deux types de relations entre les cas d'utilisation : l'inclusion et l'extension. Un cas d'utilisation peut être compris dans un autre cas d'utilisation ; ceci signifie que le service spécifié par le deuxième est compris dans le service du premier. Un cas d'utilisation peut étendre un cas d'utilisation père ; ceci signifie que le service du premier est une spécialisation du service du deuxième. Les relations entre les cas d'utilisation sont notées comme des flèches de dépendances avec les mots-clés <<include>> pour l'inclusion et <<extend>> pour l'extension.

Donc le modèle des cas d'utilisation et le modèle orienté-but altéré sont deux entrées pour le processus de construction du modèle de cas de changement Figure 4.13.

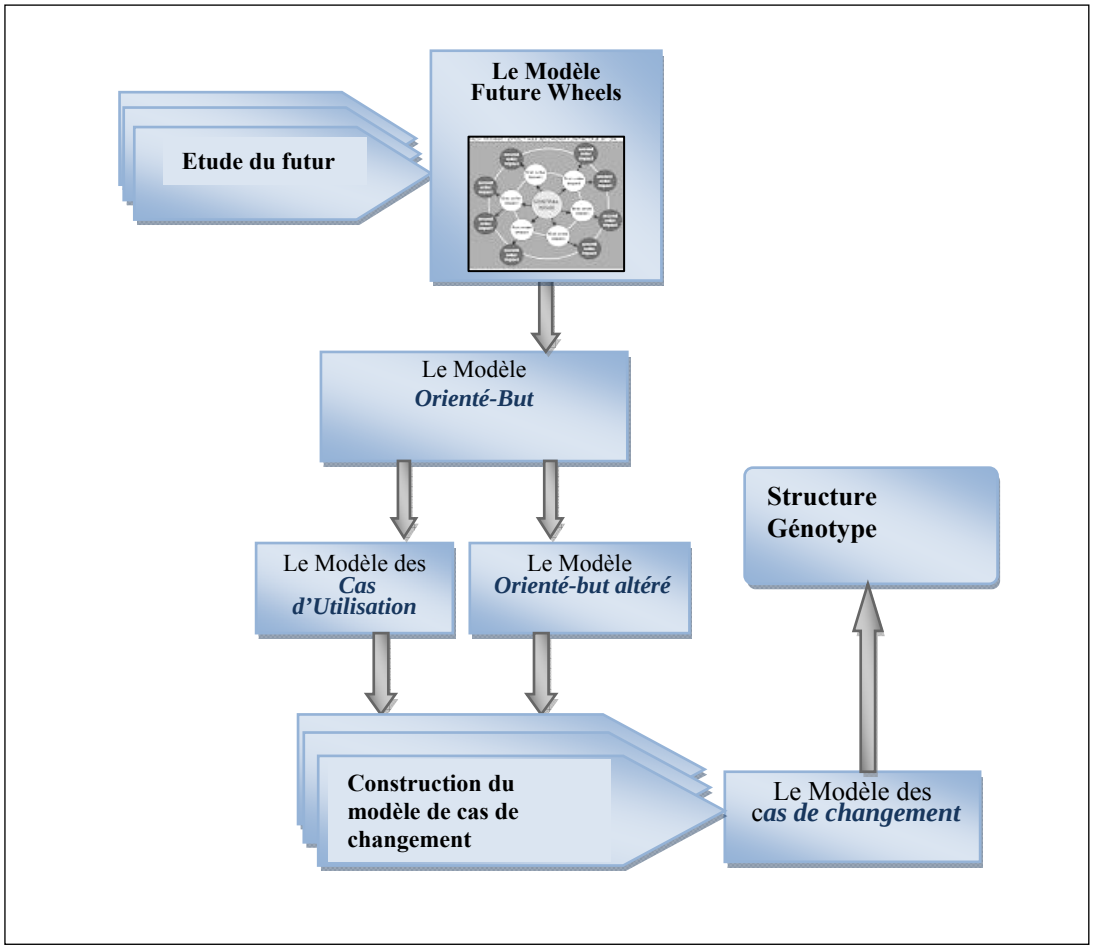


Figure 4.12 Processus de construction des cas de changement

En effet la figure 4.12 résume le processus d'ingénierie des besoins d'évolution anticipé proposé dans l'approche. Etant donné un modèle de cas d'utilisation existant, ou quasi-existant si on parle au sens RUP, une étude pour extraction des besoins d'évolution futurs est établie en utilisant la méthode « Futures wheel ». L'élicitation des évènements futurs est représenté à l'aide du modèle Futures wheel. Il contient l'ensemble des conséquences, résultat des évènements et tendances futurs. Le modèle de Roue des Futurs est une entrée

au processus d'altération du modèle de buts. Le modèle des buts altérés et le modèle des cas d'utilisation sont des entrées au processus de création des cas de changement.

4.6 Les Activités de l'Analyse de l'Impact et la Traçabilité

Pour l'utilisation efficace du modèle de cas de changement, il faut également l'appui méthodologique de la traçabilité d'un niveau de la conception à l'autre [Eck 96]. La traçabilité des exigences est définie comme la capacité à décrire et suivre la vie d'une exigence dans les deux directions, vers son origine, ou vers sa mise en œuvre, en passant par toutes les spécifications connexes [Soit 05]. La traçabilité a le plus de valeur dans une méthodologie lorsque les liens sont bidirectionnels [Eck 96]. La traçabilité comme préconisé dans [Eck 96] est à deux niveaux : Premièrement, du changement aux différents cas de changement qui le représente. Deuxièmement, du cas de changement aux cas d'utilisation qu'il affecte. La méthodologie de développement doit prendre en charge la capture et le maintien de liens de traçabilité entre les différentes phases du processus de développement. Depuis cette époque, de nombreuses techniques différentes ont été utilisées pour représenter les relations de traçabilité, y compris les approches classiques telles que les matrices, les bases de données, des liens hypertextes, les approches à base de graphes, les méthodes formelles, et les systèmes dynamiques [Aiz 06]. L'outil CASE d'IBM RequisitePro [IBM 12], a présenté des solutions de traçabilité les plus avancées, qui prennent en charge la gestion de la traçabilité des informations validité de la surveillance des changements d'éléments liés et indiquant des liens suspects.

Le développement d'un produit logiciel est en fait l'exécution d'un processus par un ensemble d'agents, dont certains sont des êtres humains et d'autres des outils [Li 09]. Dans le cadre de l'approche proposée, nous préconisons l'utilisation des outils CASE plus élaborés, à savoir soutenir la traçabilité des changements aux cas de changement de représentation et de chaque cas de changement dans les cas d'utilisation concernés. Ainsi, l'impact d'un changement au niveau du cas d'utilisation (c.à.d. l'accent du changement) est une approximation de l'impact potentiel du changement.

La décision difficile à prendre est de déterminer combien et quels changements potentiels sont suffisamment importants pour justifier d'un examen lors de la conception initiale du système. C'est une question clé pour les coûts et les contraintes sur les délais de livraison.

4.7 La Structure Génotypique

Le génotype comprend l'implémentation des évolutions anticipées et non anticipées. Le modèle des cas de changement anticipés est implémenté durant les deux phases Elaboration et Transition après une étude sur l'impact de changement. L'implémentation se base sur le modèle d'analyse et de conception déjà élaborés. Il s'agit d'une re-conception [Eck 96]. Le résultat de cette re-conception est nouveau modèle d'implémentation adapté relié au modèle initial. Le RUP est un processus de développement basé sur le formalisme UML. La figure 4.13 représente graphiquement les différents modèles produits par les workflows ou disciplines du RUP, allant du modèle des cas d'utilisation jusqu'aux diagrammes de déploiement.

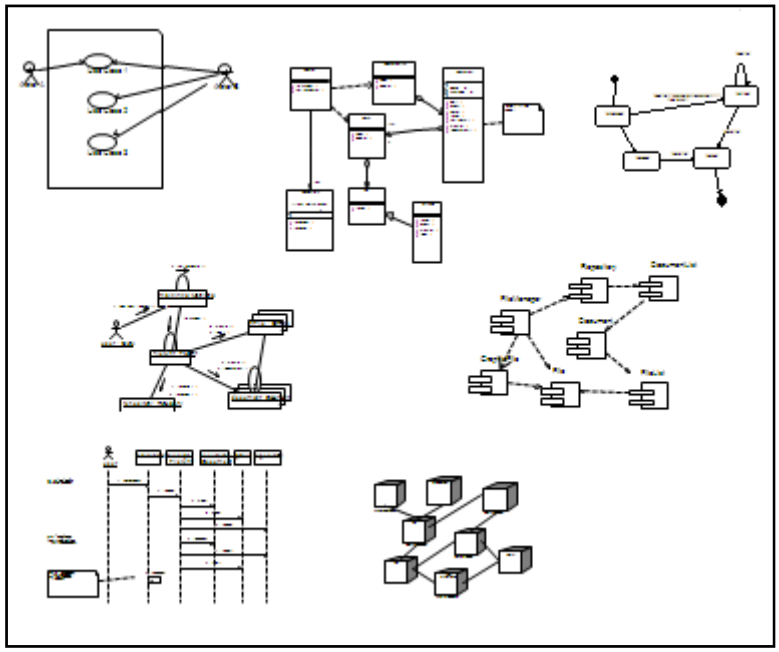


Figure 4.13 Différents diagrammes UML utilisés durant le processus de développement

La structure du génotype à un état I est représentée dans la figure 4.14. Cette structure est composée de plusieurs vues du système. Chacune des vues est liée à l'autre par un lien de traçabilité particulier ayant une structure spécifique (exemple lien hypertexte).

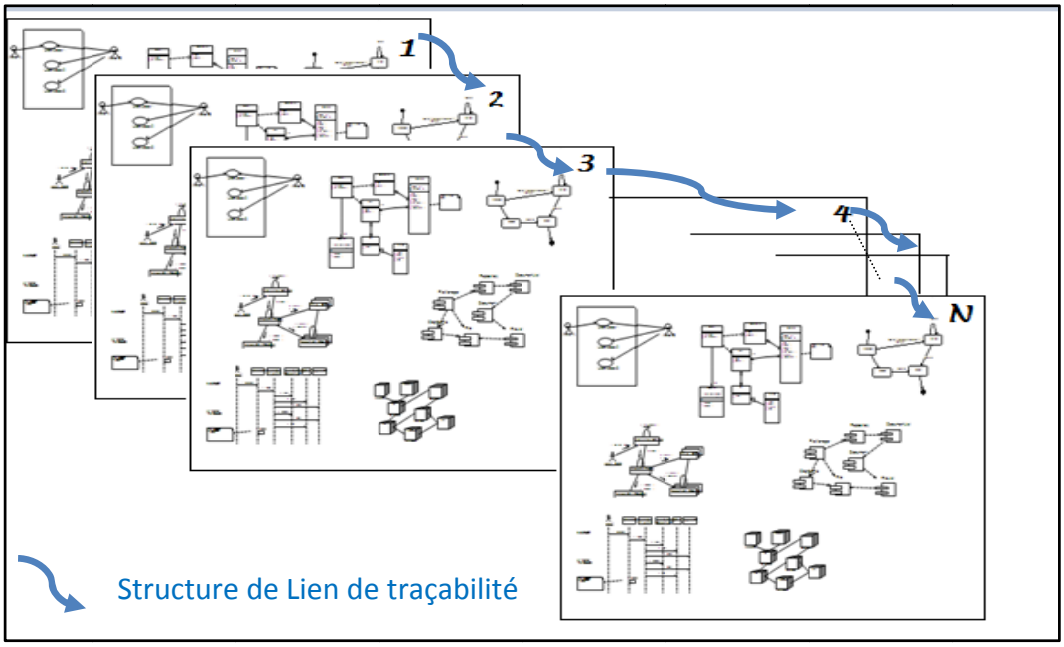


Figure 4.14 Différents vues de la structure génotypique

4.7.1 Réalisation Cas d'Utilisation

Une « Réalisation cas d'utilisation » décrit comment un cas d'utilisation particulier est réalisé dans le modèle de conception, en termes d'objets collaborateurs [IBM 15]. L'objectif d'une réalisation de cas d'utilisation est de produire une conception orientée objet bien structurée pour la mise en œuvre de comportement défini dans le cas d'utilisation. L'ensemble de Réalisation de cas d'utilisation représente un moyen de regrouper un diagramme de classes et des diagrammes de séquences ou de collaboration. On retrouvera dans le diagramme de classes les classes qui mettent en œuvre le cas d'utilisation associé au « use case realization » (structure des classes et relations entre classes) (Figure 2015). On retrouvera dans les différents diagrammes de séquences (ou de collaboration) une documentation des différents évènements échangés entre les objets afin de réaliser les différents scénarios décrit dans le cas d'utilisation.

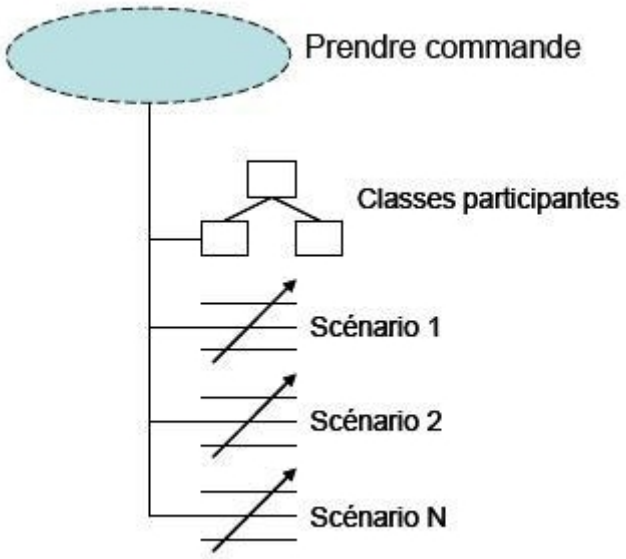


Figure 4.15 Réalisation de cas d'utilisation

La notion de réalisation est d'une importance cruciale car elle permet de répondre aux besoins de traçabilité entre les exigences que sont les cas d'utilisation et la solution apportée pour répondre à ces exigences (i.e. les classes du modèle). Un cas de changement raffine un cas d'utilisation en ajoutant une relation entre le cas de changement et les cas d'utilisation qui sont affectés par le changement [Eck 96]. Ceci facilite l'implémentation des cas de changement par modification, réutilisation et adjonction de scénario existant dans la réalisation de cas d'utilisation affecté par le changement.

4.7.2 Lien entre Cas de Changement et Cas d'Utilisation

Le lien ou la relation entre le cas de changement et l'ensemble des cas d'utilisation affectés par le changement reste une heuristique pour la conception et l'implémentation des cas de changement. Cette notion est une perspective de recherche très importante pour l'approche proposée, méritant, plus d'effort et contribution pour résoudre les aspects de mise en œuvre d'Onto-RUP.

4.8 La Discipline : Unant-Evo

Les changements/évolutions non anticipées se produisent après la livraison du système logiciel. Actuellement, la maintenance des logiciels et de l'évolution est une zone relativement sous-étudiée en tenant compte de ses effets sur les coûts. Le RUP contient déjà tout ce qui est nécessaire en termes de rôles , des activités, des artefacts et des lignes

directrices pour couvrir la maintenance d'une application logicielle, et en raison de la nature essentiellement itérative du RUP, la capacité à évoluer, corriger ou affiner des artefacts existants est inhérente à la plupart de ses activités [Kru 01, Erd 03]. Dans le contexte des systèmes ontogénétiques le changement est un processus continu. Ainsi, l'entretien ne peut être ajouté à toutes les disciplines existantes, une discipline nouvelle et dédiée à la maintenance doit être établie. Nous proposons la discipline de l'évolution non anticipée pour promouvoir, assurer et préserver la maintenabilité, diminuer les efforts de maintenance après livraison et automatiser des tâches humaines. La discipline contient des activités pour faciliter le développement continu du système et sa mise à jour dynamique. Une évolution peut invalider un produit existant, même si plusieurs techniques de préparation ont été incluses. Mais en faisant un choix judicieux de mécanismes d'extension et de techniques d'adaptation ainsi que d'une bonne conception de l'architecture initiale, il devrait être possible de rendre les systèmes logiciels plus flexibles à l'évolution [Kni 02]. Le système ontogénétique produit sera mis à jour dynamiquement quand une évolution imprévue se produit. Le processus de mise à jour de logiciel est constitué de deux parties. Tout d'abord la mise à jour est mise en œuvre, puis la mise à jour est installée. L'exigence minimale de mise à jour logicielle dynamique est la capacité du langage de programmation hôte à charger un nouveau code à l'exécution. Cette fonctionnalité est disponible dans la plupart des langages de programmation modernes. La discipline doit être guidée par un ensemble d'outils. Par exemple : Changer les cas, analyser l'impact, traçabilité, gestion des versions, documentation, etc.

Conclusion

Dans ce chapitre, nous avons présenté Onto-RUP, une démarche pour le développement des systèmes ontogénétique. L'approche propose deux disciplines supplémentaires au RUP : la première étant « Ant-Ev », concerne des activités d'élicitation et modélisation des évolutions anticipées, tandis que « Unant-Ev », concerne l'évolution non anticipée. Pour l'élicitation des besoins d'évolution anticipée, l'approche utilise une combinaison de plusieurs modèles : Futures wheel issu de la sociologie, modèles orientés-buts issus des GORE, cas d'utilisation d'UML et le formalisme de cas de changement. Nous donnons dans le chapitre suivant des exemples d'application pour illustrer la démarche proposée.

CHAPITRE 5
VALIDATION PRATIQUE

L'un des moyens possibles permettant d'évaluer les modèles de processus ou méthodologie de développement de système logiciels est de choisir des systèmes exemples et employer la démarche pour le développement de ces exemples d'application. Nous proposons comme exemple d'application " le système de gestion des transactions bancaire".

5.1 La Démarche Proposée à Travers une Etude de Cas : Gestion des Transactions Bancaires

En utilisant ce système, les clients seront en mesure d'obtenir de l'argent en espèces en utilisons leurs cartes bancaires aux distributeurs de billets. Nous nous concentrons dans ce cas sur la première phase de développement, ***Inception,*** de la démarche Onto-RUP.

5.1.1 Les Caractéristiques du Projet

Nous supposons caractéristiques suivantes pour notre projet bancaire:

- Toute nouvelle demande de l'équipe de développement
- Les clients externes
- Équipe de 5 développeurs y compris le gestionnaire et analyste futuriste
- Le chef de projet est l'architecte
- 15 cas d'utilisation

5.1.2 Le Nombre d'itérations

La description du produit initial a été créée au cours de la soumission et proposition du projet : Petite application, facile à comprendre.

Alors que la phase d'inception nécessite une seule itération, Onto-RUP nécessite plus d'une itération. Dans la première itération une bonne partie est consacrée à l'analyse des besoins de l'application. Tandis que la deuxième itération concerne l'extraction et l'analyse des évolutions anticipées des besoins.

5.1.3 Liste d'activités avec un plan d'itération

Selon RUP [Kru 00] la planification est une décomposition de processus, autrement dit ce qu'il faut réaliser pour atteindre certains objectifs en un temps donné. C'est là qu'on peut observer en action les concepts de **phases** et d'**itérations**, ainsi que les **jalons** majeurs et mineurs qui les accompagnent.

Un plan d'itération est un plan à grain fin, cadré dans le temps [Kro 03]. Comme il y a un plan par itération, il s'étend sur une durée limitée, suffisante pour permettre aux membres de l'équipe de percevoir le type de planification détaillée dont ils sont coutumiers, avec le niveau de granularité approprié aux tâches et à l'allocation successive de ces dernières aux différentes personnes. Le plan d'itération est construit à l'aide des techniques et outils de planification habituels (diagrammes de Gantt, etc.).

Le développement d'un **plan** d'**itération** a quatre étapes :

1. Déterminer la portée de l'itération
2. Définir le critère d'évaluation de l'itération.
3. Définir les activités de l'itération.
4. Attribuer des responsabilités.

Les deux tables 5.1 et 5.2 représentent respectivement la liste d'activités de la première et deuxième itération de la phase Inception.

		Activité
Iteration N°1	**Objectif 1** (Compréhension)	Ecrire le document vision (4 pages)
		Faire le brainstorming initial des cas d'utilisation et passer 1 heure sur chacun pour le décrire.
		Meeting de l'équipe pour consolider les cas d'utilisation
		L'équipe passe un autre jour pour détailler les cas d'utilisation critiques.
	Objectif 2 (fonctionnalités clés)	L'architecte sélectionne 5 cas d'utilisation qui soient les plus critiques et les discute avec le client.
		Meeting de l'équipe, l'architecte explique pourquoi les cas d'utilisation retenus sont les plus critiques.
		L'architecte documente les cas d'utilisation sous forme d'une spécification logicielle initiale des exigences.
	Objectif 3 (Anticipating Changes/ Evolutions)	Meeting équipe+futuristes (étude Futures wheel , Comprendre les liens entre les événements, les tendances et les actions, agréer les évolutions rationnelles.
		Modèles but adapté
		Eliciter les cas changement
		Construction des modèles des cas de changement
	Objectif 4 (Solution	Travailler (1 jour) sur un prototype fonctionnel

	potentielle)	Acquérir ou implémenter certains éléments clés de l'architecture (1 jours)
	Objectif 5 (Risques)	Evaluer risques et coûts (chef projet)
	Objectif 6 (Processus)	Meeting de 1 heure de l'équipe pour discuter comment les membres doivent travailler. Le chef de projet/architecte créera ensuite une description du processus et des règles liées à la création de l'itération (les tâches à effectuer, artefacts produits, les modèles à utiliser, comment documenter)

Table 5.1 Liste d'activités avec plan d'itération N°1

Les efforts additionnels concernant de l'anticipation des besoins d'évolution/changement nécessite une deuxième itération pour la phase d'Inception du cycle de développement (Table 5.2).

Iteration N°2		**Activité**
	Objectif 1 (Compréhension)	Meeting de l'équipe pour raffiner le document vision en considérant les évolutions anticipées.
	Objectif 2 (Fonctionnalités clés)	L'équipe affine le modèle des cas d'utilisation
	Objectif 3 (Anticiper Changements / Evolutions)	L'équipe crée les traces (liens de traçabilité)
		L'équipe affine le modèle des cas de changement
	Objectif 4 (solution potentielle)	Prototype fonctionnel total besoins
		L'équipe passe 1 jour pour faire le portrait des alternatives possibles du futur.
	Objectif 5 (Risques)	Etudier risques de prise en charge des évolutions
	Objectif 6 (Processus)	Meeting de l'équipe d'une ½ journée pour discuter comment ils doivent travailler.
		Chef projet / architecte crée ensuite une description du processus futur, discute des risques, les coûts et décide de l'adoption / non adoption de la mise en œuvre des changements anticipés. Si la décision est «oui», alors il documente et précise les outils nécessaires.

Table 5.2 Liste d'activités avec plan d'itération N°2

Une décision doit être prise à la fin de la phase Inception sur le jalon "Objectif cycle de vie". La phase est soit annulée ou répétée après avoir été repensée pour répondre à certains critères. Selon l'approche ONTO-RUP, les deux artéfacts les plus importants de la phase Inception sont les deux modèles respectivement des cas d'utilisation et des cas de changement.

5.1.4 Les Activités du processus d'élicitation des besoins

5.1.4.1 Construction du modèle des buts

La première étape pour l'analyse orientée-buts est de représenter les parties prenantes de l'organisation et leurs dépendances à l'aide d'un diagramme d'acteurs représentant les agents, les rôles ou les fonctions au sein de l'organisation. Cela commence par un très haut niveau d'analyse visant à exprimer les responsabilités et les relations impliquant les différentes composantes d'une organisation. Ensuite, l'analyse procède avec plus de détails par la décomposition des acteurs de haut niveau dans des sous-acteurs et se termine avec l'identification d'agents responsables d'une seule activité.

Les parties prenante sont multiples, le client, la banque, client-ATM, gestionnaire financière, etc. On donne dans la figure 5.1 le modèle initial des buts pour un client utilisant le DAB. Le but principal est Trait-DAB, qui consiste en une conjonction de deux sous-but. Le premier est l'Identification du client qui est un but non fonctionnel (softgoal) lié à l'assurance de sécurité. Le deuxième sous-but est l'Authentification, composée de la tâche Accès-autorisé qui est à son tour composés de la tâche Carte à puce. Cette dernière est à son tour composée de différentes tâches traitées par le système comme Retrait d'argent, Afficher historique du compte, etc.

Pour le contexte du système étudié, nous nous intéressons aux acteurs Client-DAB à qui le système offre des possibilités fonctionnelles pour effectuer des transactions en utilisant le DAB, telle que Retrait d'argent, Consultation du compte, etc.

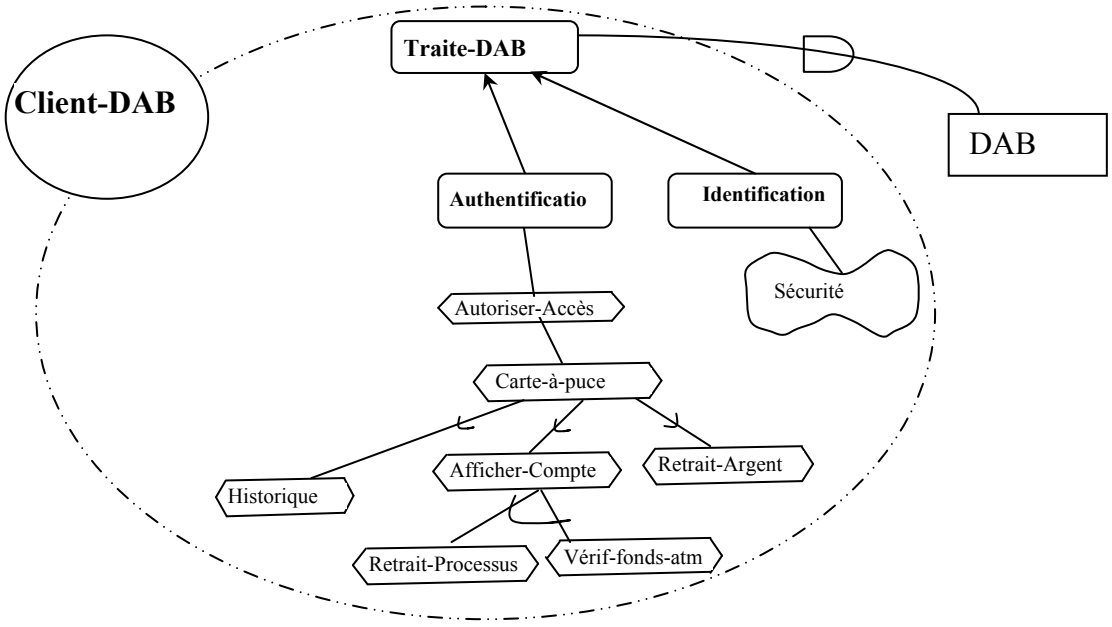


Figure 5.1 Le modèle initial des buts « Client-DAB »

5.1.4.2 Construction du modèle des cas d'utilisation

Les cas d'utilisation décrivent l'interaction entre un système et son environnement. Les cas d'utilisation sont des boîtes noires qui décrivent la réaction du système dans sa globalité, sans indiquer comment le système atteint l'objectif visé dans une utilisation donnée. Un ensemble complet de cas d'utilisation spécifie toutes les possibilités d'utilisation d'un système et, par conséquent, décrit tout le comportement requis du système. Le modèle des cas d'utilisation adopté par Onto-RUP présente la spécification formelle des besoins utilisateurs. Ce modèle sépare le système en acteurs et cas d'utilisation.

La table 5.3 donne un exemple de cas d'utilisation montrant comment s'effectue une opération d'authentification automatique d'un client lors d'une transaction bancaire.

Use case name	Authentification automatique d'un client
Context of use	A tout moment, les clients titulaires de comptes au niveau de la banque et qui disposent de cartes bancaires valides, peuvent initier des transactions sur leurs comptes et ces transactions nécessitent l'authentification des clients
Main Actor	Le client
Preconditions	Le client dispose d'une carte et souhaite effectuer une transaction
Postconditions	Le client est authentifié
Used use cases	Aucun
Description	1: Le client introduit sa carte bancaire 2: Le système vérifie la validité de la carte 3: Le système invite le client à introduire son code personnel. Le client introduit le code demandé 4: Le système vérifie la validité du code 5: Fin
Alternatives	2a: Si la carte est invalide, lancer un bip sonore, informer le client, éjecter la carte puis aller à l'étape 5 4a: Si le code est invalide aller à l'étape 3. Au bout de trois tentatives informer le client, éjecter la carte et aller à l'étape 5

Table 5.3 Cas d'utilisation relatif à l'authentification automatique d'un client

La table 5.4 donne le cas d'utilisation du retrait bancaire via le DAB.

Use case name	Retrait via DAB
Context of use	La banque dispose d'une multitude de DAB installés dans, ou à proximité de, ses agences. Les clients peuvent lancer une opération de retrait qui leur permet d'obtenir des billets de banque
Main Actor	Le client
Preconditions	Le client est authentifié et souhaite effectuer un retrait
Postconditions	Le solde du client est inchangé ou diminué du montant retiré. Le solde est positif ou nul
Used use cases	UC1: Arrêt d'interaction avec un client
Description	1: Le système invite le client à introduire un montant pour le retrait. Le client introduit le montant 2: Le système vérifie que le montant est supérieur ou égal au solde et délivre les billets 3: Arrêt d'interaction avec UC1
Alternatives	2a: Si le montant est supérieur au solde, informer le client et aller à l'étape 2. Au bout de trois tentatives, aller à l'étape 3

Table 5.4 Cas d'utilisation relatif à une opération de retrait de billets

Le modèle global des cas de changement est montré en figure 5.2.

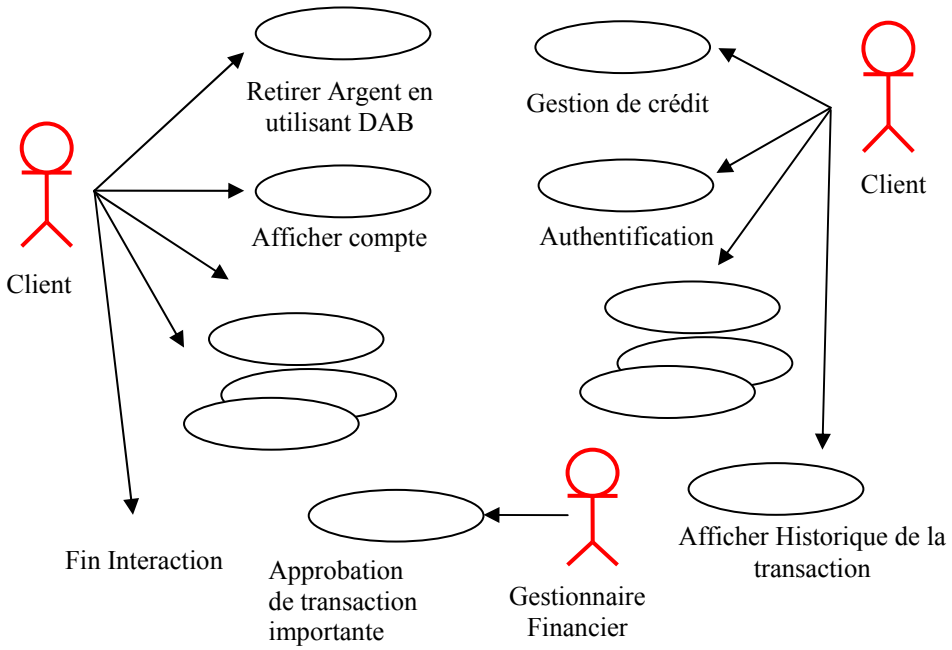


Figure 5.2 Le modèle global des cas d'utilisation

5.1.4.3 Construction du modèle Futures wheel

L'extraction des tendances et évènement futurs s'est focalisé sur deux évènements ayant un impact stratégique et organisationnel intéressant :

- Utilisation de nouvelles technologies pour l'authentification des clients « Utilisation des dispositifs biométriques ». Illustré dans la figure 5.3.
- Fidélisation des anciens clients « Solde négatif ». Illustré dans la figure 5.4.

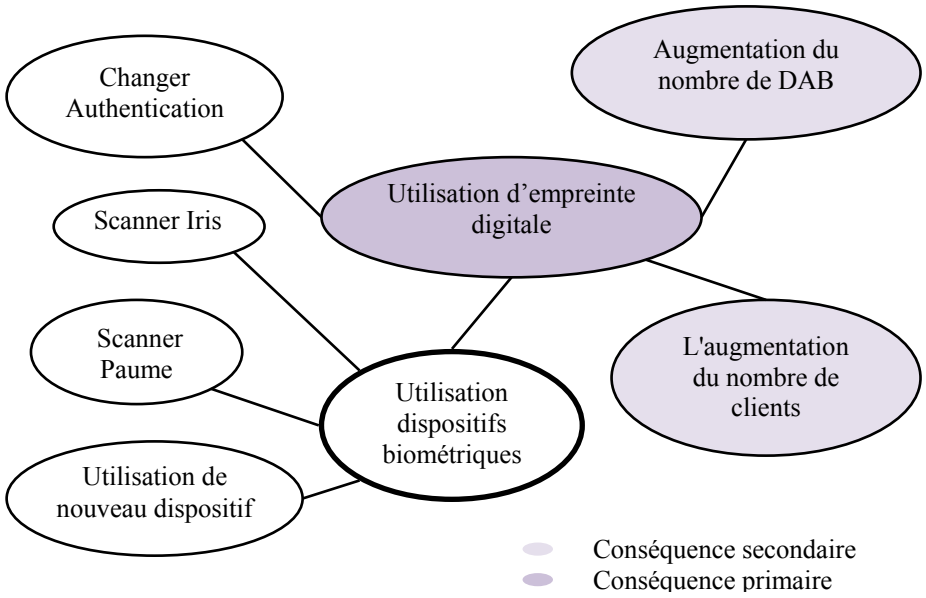


Figure 5.3 Modèle futures wheel : évènement utilisation des dispositifs biométriques

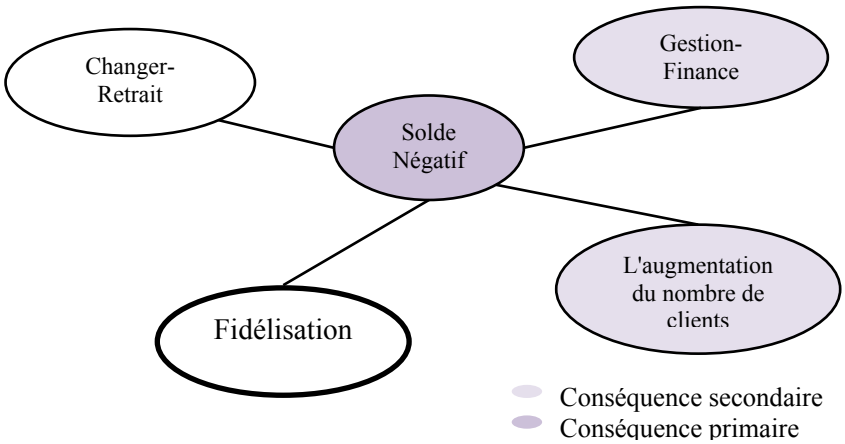


Figure 5.4 Modèle Roue des futurs : évènement fidélisation du client

5.1.4.4 Construction du modèle de buts altéré

La prise en compte des évènements élicités pendant les séances Futures wheel permet d'altérer le modèle initial des buts (Figure 5.1), en ajoutant des buts, tâches, etc. comme illustré dans la figure 5.5.

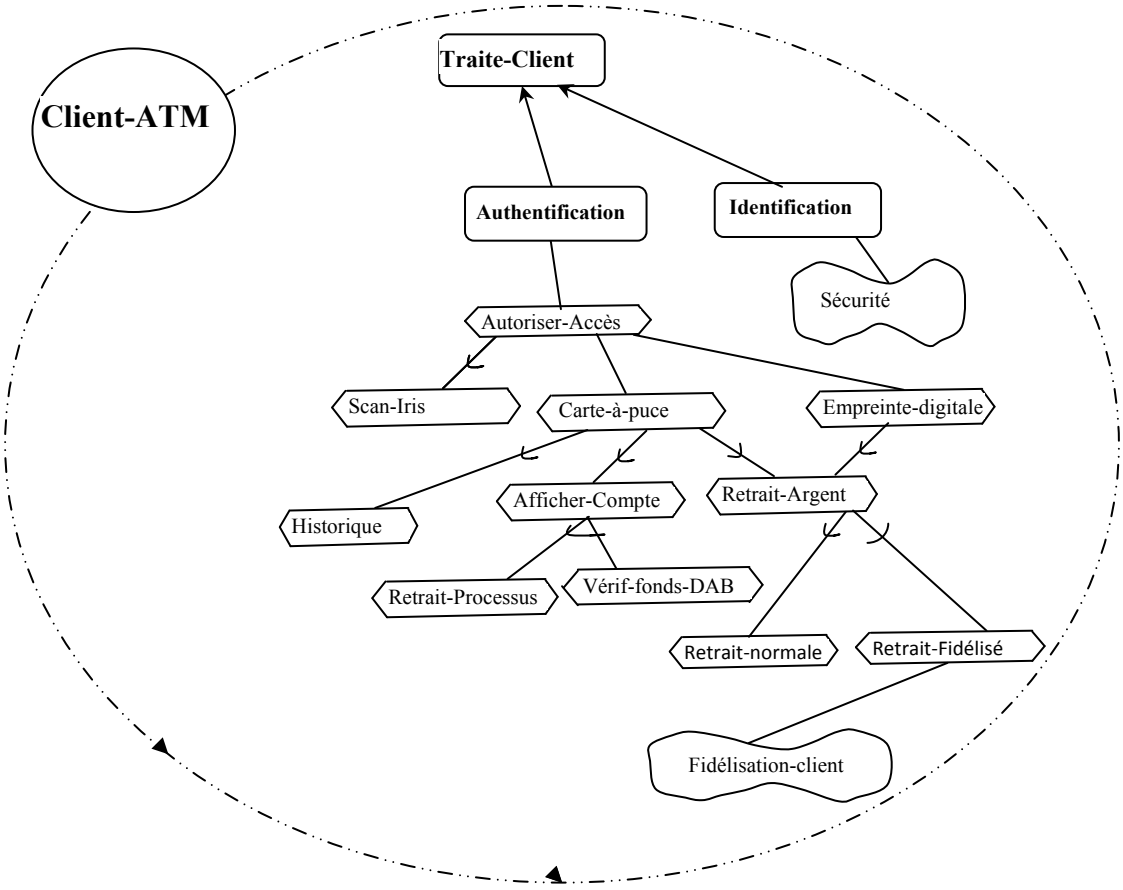


Figure 5.5 Modèle altéré des buts

5.1.4.5 Construction du modèle de cas de changement

Le modèle des cas de changement se construit en analysant les deux modèles :

- Modèle des cas d'utilisation et
- Modèle altéré des buts

Nous donnons dans les deux tables 5.5 et 5.6 les deux cas de changements.

Change case name	Retrait sans solde pour client fidèles
Occurrence context	La banque souhaite devenir plus compétitive en fidélisant ses anciens clients
Concerned use cases	Retrait via le DAB. Ce cas d'utilisation reste valable
Reasons behind the change	Nouvelle règle de gestion visant à fidéliser les clients
Likelihood	Le changement est certain et s'opérera pour les clients affiliés à la banque depuis deux années ou plus
Nature of the change	Permanent
Scope of the change	Touche seulement les clients affiliés depuis deux ans ou plus. Tous les cas d'utilisation utilisant Retrait via DAB, doivent utiliser le nouveau cas d'utilisation lorsqu'il s'agit de compte datant de deux ans et plus
Triggering factor	Deux ans après l'ouverture d'un compte
Context of use	La banque dispose d'une multitude de DAB installés dans ou à proximité de ses agences. Les clients peuvent lancer une opération de retrait qui leur permet d'obtenir des billets de banque
Main Actor	Le client
Preconditions	Le client souhaite effectuer un retrait et est authentifié
Postconditions	Le solde du client est inchangé ou diminué du montant retiré. Le solde est supérieur ou égal à -200
Used use cases	**UC1:** Arrêt d'interaction avec un client
Description	1: Le système invite le client à introduire un montant pour le retrait. Le client introduit le montant 2: Le système vérifie que le montant est inférieur ou égal à 'solde+200' et délivre les billets 3: Arrêt d'interaction avec UC1
Alternatives	2a: Si le montant est supérieur à 'solde+200', informer le client et aller à l'étape 1. Au bout de trois tentatives, aller à l'étape 3

Table 5.5. Cas de changement relatif au retrait effectué par les clients fidèles

Change case name	Ajout d'une procédure d'authentification
Occurrence context	La banque renforce ses dispositifs d'authentification par des dispositifs biométriques fiables et d'utilisation simple
Concerned use cases	Authentification automatique d'un client. Le cas d'utilisation reste valable mais peut être substitué par le nouveau à tout moment et en toute circonstance
Reasons behind the change	Acquisition de dispositifs d'identification des empreintes et des rétines (raison technologique) pour rendre l'authentification plus fiable (aspect non fonctionnel) et plus simple (raison fonctionnelle)
Likelihood	Le changement est certain et s'opérera à partir du 1er janvier 2006
Nature of the change	Permanent
Scope of the change	Tous les DABs seront équipés du dispositif d'authentification biométrique. Tous les cas d'utilisation nécessitant l'authentification automatique du client pourront se faire aussi par authentification biométrique
Triggering factors	Dès l'installation des nouveaux dispositifs et la fin de la récupération des données biométriques des clients
Context of use	La banque dispose d'une multitude de DAB installés dans ou à proximité de ses agences. A tout moment, les clients titulaires de comptes au niveau de la banque peuvent initier des transactions sur leurs comptes et ces transactions nécessitent l'authentification des clients
Main Actor	Le client
Preconditions	Le client souhaite effectuer une transaction
Postconditions	Le client est authentifié
Used use cases	Aucun
Description	1: Le client met l'index sur le dispositif de capture d'empreinte. Le système saisit les caractéristiques de l'empreinte 2: Le système identifie, selon les caractéristiques, le client concerné 3: Fin
Alternatives	1a: Le client regarde en face (sans lunettes) une caméra qui saisit l'image de sa rétine. Le dispositif d'identification extrait les caractéristiques 2a: Si le système n'arrive pas à identifier le client, il lui demande de refaire l'opération de l'étape 1. Au bout de trois tentatives, le système informe le client qu'il n'arrive pas à l'identifier puis va à l'étape 3

Table 5.4 Cas de changement relatif à l'authentification

5.2 Outils Support d'Onto-RUP

La production d'un logiciel est une tâche très complexe mettant en œuvre plusieurs ressources, notamment humaines, matérielles et technologiques. L'automatisation des parties du processus de développement permet d'aider les développeurs dans leurs tâches en réduisant considérablement le temps de réalisation des artéfacts tout au long du cycle de vie du système ontogénétique.

Nous citons dans ce qui suit quelques outils qui sont à la fois importants et complémentaires à la démarche Onto-RUP :

- AGL qui aide le développeur à l'analyse et la traçabilité des cas de changement. Nous avons réalisé une première version d'une partie de cet outil dans [Ben 13].
- Conception et implémentation des cas de changement par une approche basée composant (Projet en perspective).

5.2.1 ORC Un outils case pour la gestion des cas de changement

Comme pour toute méthode de développement de système logiciel, un AGL est nécessaire pour Onto-RUP afin d'améliorer la vitesse et la qualité du travail de développement chaque fois que cela est possible.

ORCTOOL [Ben 13] est une partie de l'outil AGL que nous préconisons pour ONTO-RUP**.** Il vise à aider le développeur dans l'utilisation de ce formalisme durant le cycle de vie du système logiciel.

Notre travail est dédié principalement à aider le développeur dans la conception des diagrammes de cas d'utilisation tout en incorporant des cas de changement et en permettant l'ajout d'acteurs et de relations.

5.2.1.1 Objectif de l'AGL

L'objectif de l'outil étant d'offrir une assistance aux développeurs pour la création et la gestion des cas d'utilisation et leurs changements anticipés et leurs impacts sur les autre cas d'utilisation.

Il permet en plus la description graphique des cas et la génération de la documentation sous forme de matrice traçabilité.

Dans ce qui suit nous présentons un aperçu de l'outil proposé.

5.2.1.2 Vue Globale de la conception d'ORCTOOL

Le développeur construit ses diagrammes comme première activité. Il aura à sa disposition ce qui suit :

- **Vue globale sur le system :** Elle donne une bonne vue du système à développer pour le développeur et pour d'autres parties prenantes.
- **Conception les cas d'utilisation altéré par les cas de changement:** Il permet de représenter les cas graphiquement.
- **Impact du changement sur les autre cas d'utilisation :** Il nous montre les liens de traçabilité pour aider le développeur a cerner l'effet du cas changement sur les cas d'utilisation.
- **Sauvegarde du projet (diagramme) et partage des ressources :** Le projet peut être enregistré avec la possibilité de modifier ultérieure si d'autres cas de changement sont élicités. En plus, il offre la possibilité de partage avec d'autres sites moyennant quelques fichiers de configuration.
- **Génération de la documentation :** On peut extraire plusieurs types d'information sur le projet, telle que la matrice de traçabilité.

La figure 5.6 donne un aperçu sur le système réalisé.

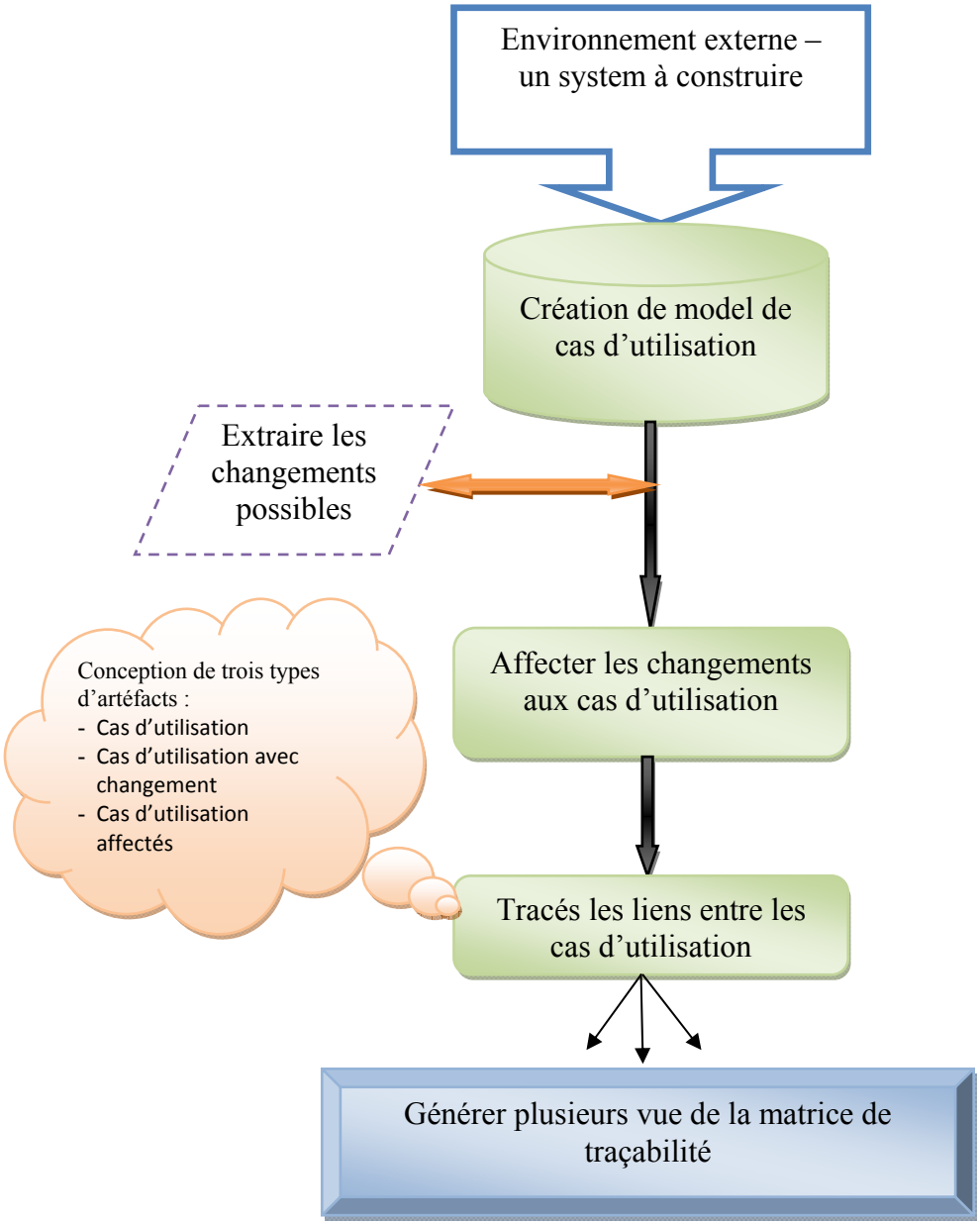


Figure 5.6 Schéma général du système proposé

5.2.1.3 Aspects Pratiques

Outils de réalisation

Nous avons utilisé dans l'implémentation de ORCTOOL, Jdom (Java Document Object Model) et Netbeans (environnement de développement intégré (EDI) . Pour cela, nous avons :

- Importer le package le « org.jdom.* » dans les classes manipulant les fichiers XML on in/output (lecture/écriture),
- Implémenter les méthodes nécessaires pour réaliser les différentes fonctionnalités et
- Importer la bibliothèque « jdom.jar » dans le Build Path du projet (figure 5.7).

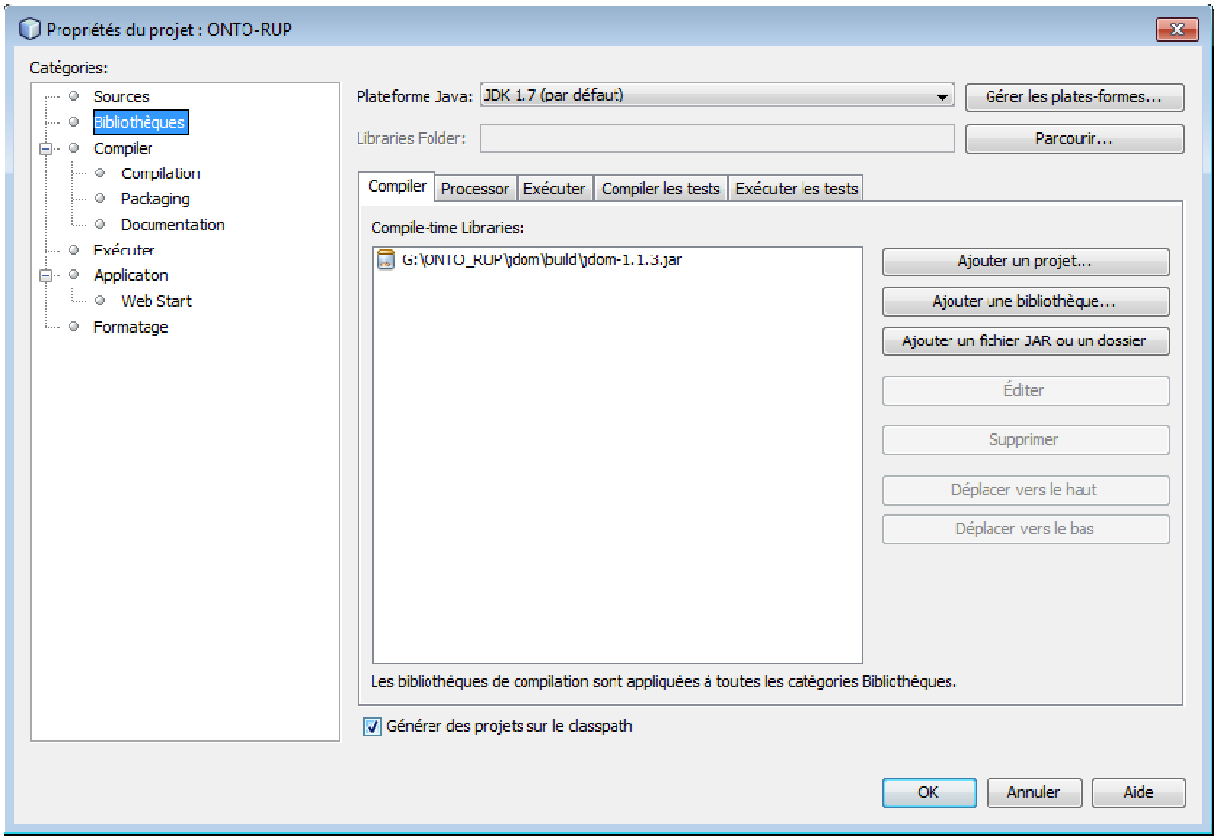


Figure 5.7 importation de la bibliothèque jdom.

Elément composants ORCTool

L'espace de travail de ORCTool se compose à 5 éléments principaux :

- **Elément N°1 (zone de dessin)** : c'est le plus important élément où seront affichés les diagrammes. La zone affiche les nouveaux cas et acteurs automatiquement sous forme d'une matrice. L'espace étant divisé virtuellement en zones.
- **Elément N°2 (barre d'outils) :** Contient des raccourcis pour réaliser les différentes tâches de l'AGL.
- **Elément N°3 (Arbre des cas d'utilisation) :** Permet d'offrir une vue globale sur les cas d'utilisation ainsi que leurs changements et facilite leur gestion.

- **Elément N°4 (zone morte) :** Elle n'est pas accessible par l'utilisateur, elle est réservée pour recevoir les cas et les acteurs lorsque la zone de dessin considérée est pleine.
- **Elément N°5 (barre menue) :** Barre classique des menus (fichier, édit, help, …).

ORCTool affiche l'écran montré en figure 5.8 lors de son lancement.

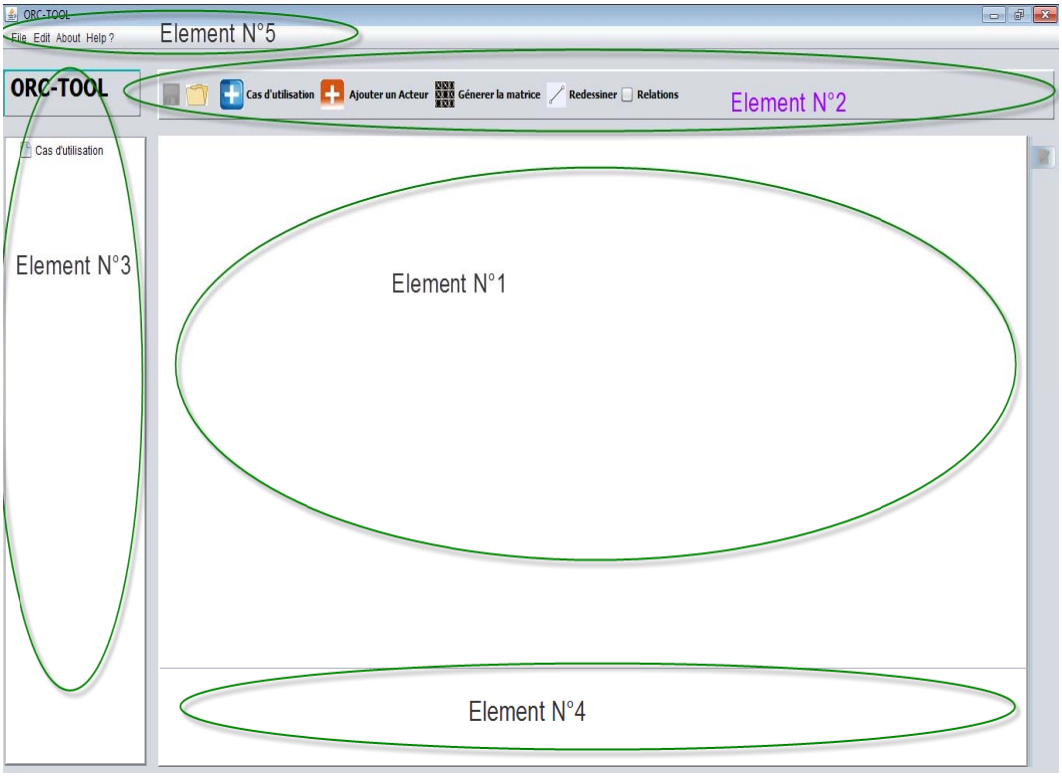


Figure 5.8 Fenêtre Principale.

Synoptique de déroulement

- On ajoute les éléments de notre diagramme (cas /acteur /relation) à l'aide de la barre d'outils, la fenêtre suivante s'affiche (Figure 5.9).

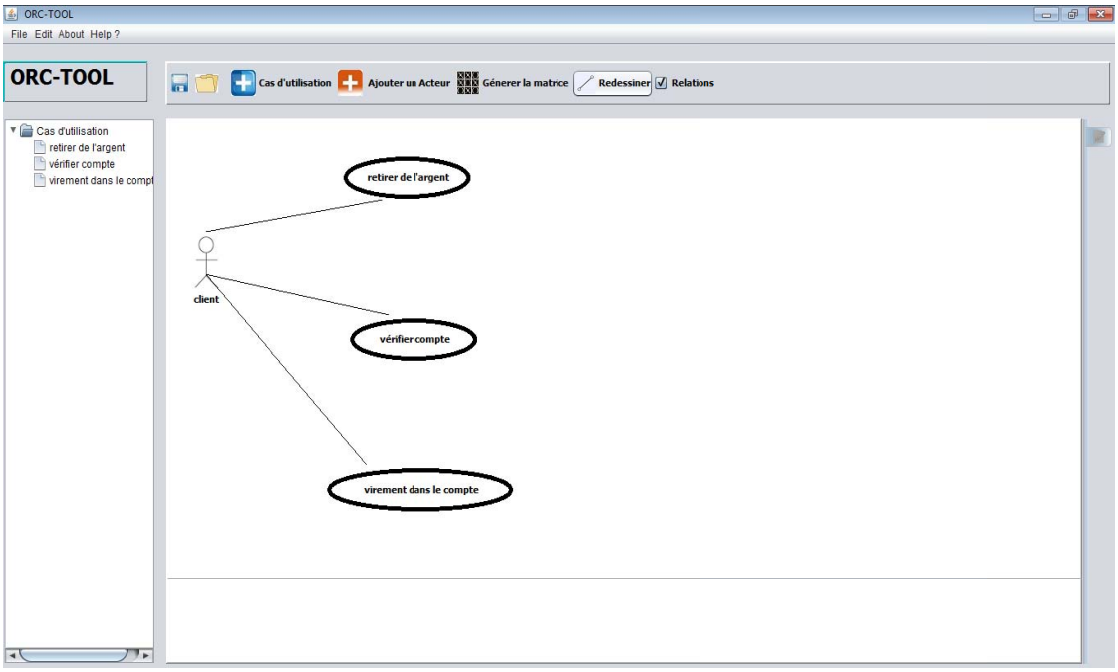


Figure 5.9 exemple d'un diagramme simple

- On indique les cas de changement à partir de l'arbre des cas d'utilisations de projet (figure 5.10)

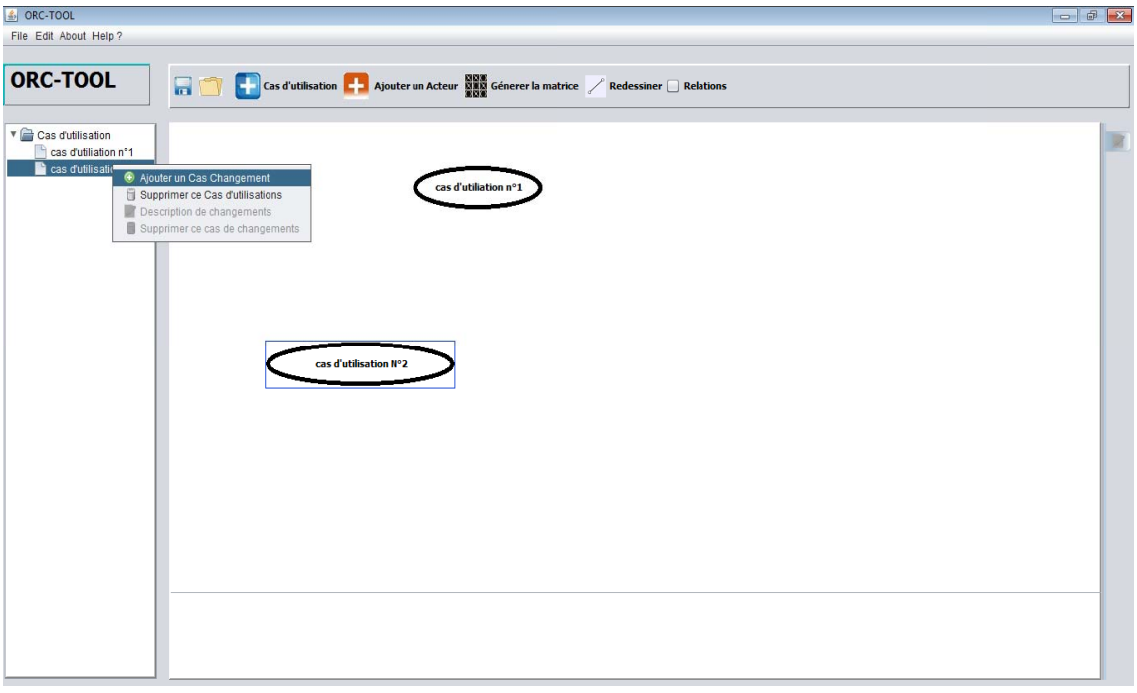


Figure 5.10 affectation de changement au cas d'utilisation

La meilleure façon d'utiliser cet AGL est de construire tous le diagramme acteur/cas (utilisation-changement) suivis par des tracés des liens entre eux (les relations).

On utilise trois couleurs différentes pour les cas d'utilisations (figure 5.11) :

- La couleur noir pour les cas d'utilisation normal.
- La couleur rouge pour cas d'utilisations qui seront changés.
- La couleur bleu pour les cas d'utilisation liée à un ou plusieurs cas de changement.

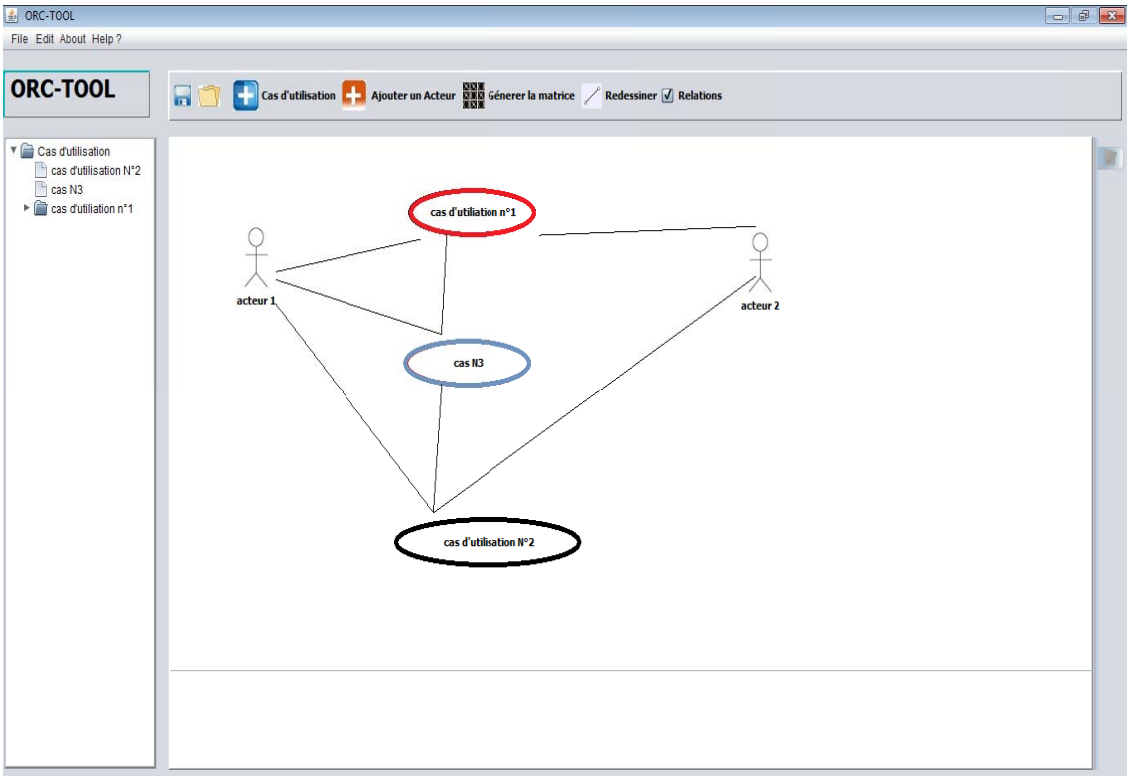


Figure 5.11 trois types de changement

A tout moment, on peut générer la matrice de traçabilité et lancer son impression (Figure 5.12).

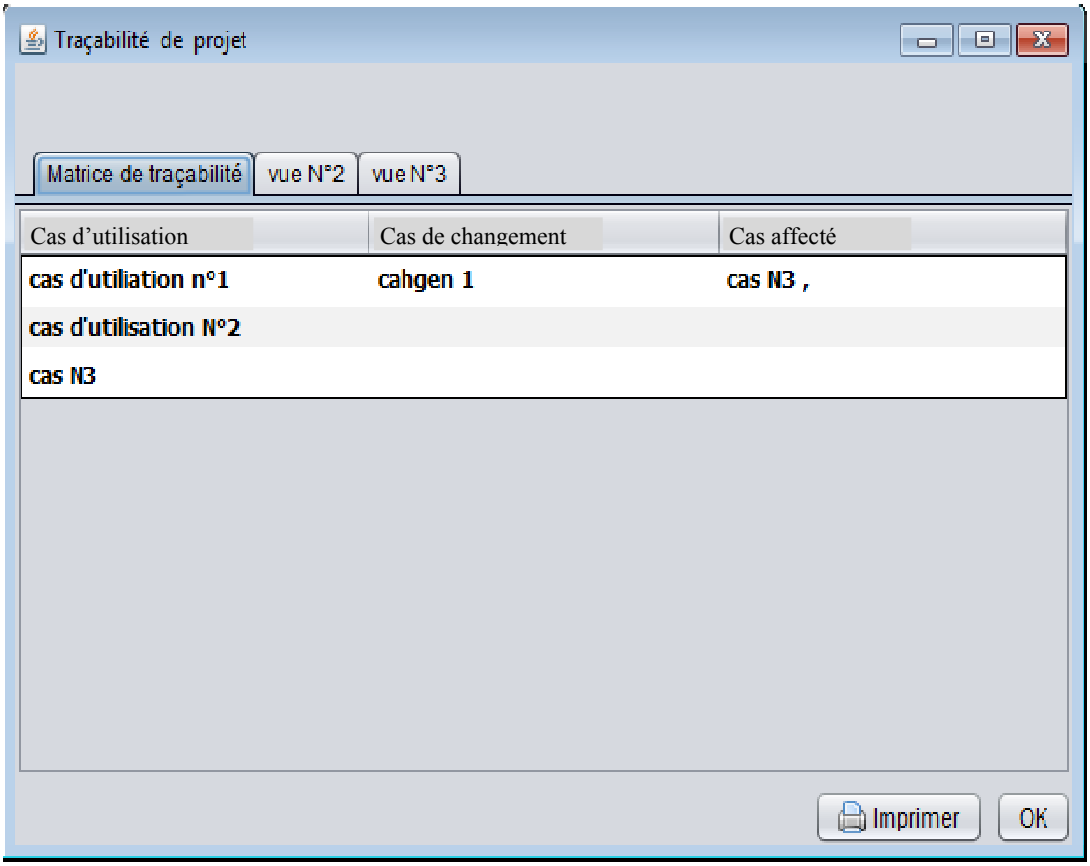


Figure 5.12 Affichage de la matrice de traçabilité et génération de document

5.2.2 Conception de cas de changement basés Composants

Le développement de logiciels à base de composants (CBSE : Component Based Software Engineering) est une approche qui vise à rendre disponibles des composants logiciels sur étagères. L'objectif est de pourvoir réutiliser des parties entières d'un logiciel et concevoir des applications logicielles par composition. Suivant cette approche, une application logicielle est construite en interconnectant plusieurs composants. Les composants sont une vision qui met en avant l'opération de composition, c'est-à-dire, assembler des composants pour réaliser un logiciel plus complexe.

5.2.2.1 Aspect de conception

Un cas de changement se traduit par un ensemble de sous composants interconnectés, qui vont remplacer les composants des cas d'utilisations concernés. Le remplacement de chaque composant par un autre composant se fera en utilisant la technique adaptée au model de composants utilisé (EJB, OSGI, FRACTAL ADL...) (Figure 5.13).

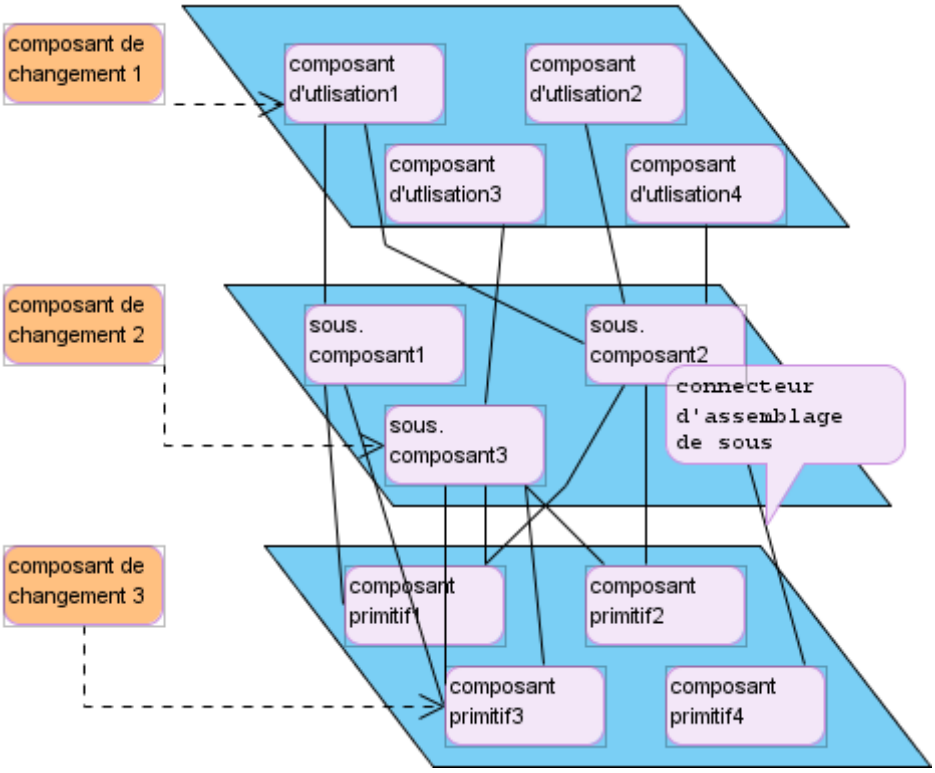


Figure 5.13 Architecture logicielle avec des composants composites

Le changement peut aussi être une nouvelle configuration du composant est sous composants qui représentent les cas d'utilisation (sans ajout de composant de changement) (Figure 5.14).

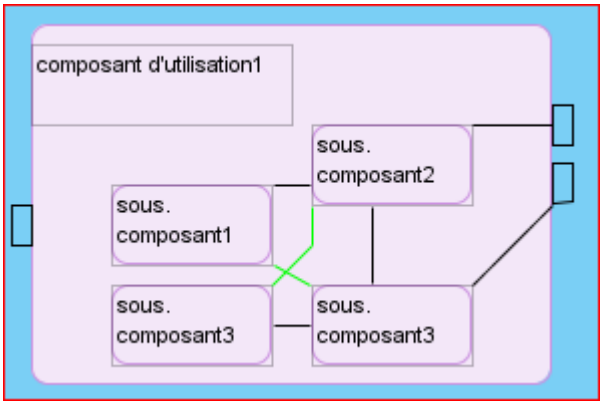


Figure 5.14 Changement interne dans un composant composite

5.2.2.2 Aspect d'implémentation

Pour implémenter l'approche proposée nous avons considéré l'application de la gestion d'une bibliothèque telle que la gestion des emprunts d'ouvrages, réservation et retour des ouvrages, etc. Les changements dans ce système consistent en l'ajout ou la suppression de quelque cas d'utilisation, ce qui aura un impact sur les diagrammes.

Chaque cas d'utilisation et cas de changement est mis dans un composant Fractal primitif comme montré dans la figure 5.15, ces composants implémentent des classes Java simples qui contiennent des méthodes (Figure 5.16).

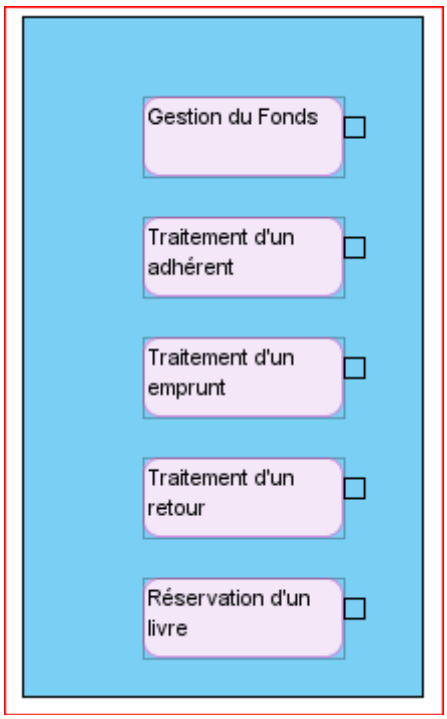


Figure 5.15 Les composants des cas d'utilisation du système de gestion d'une bibliothèque

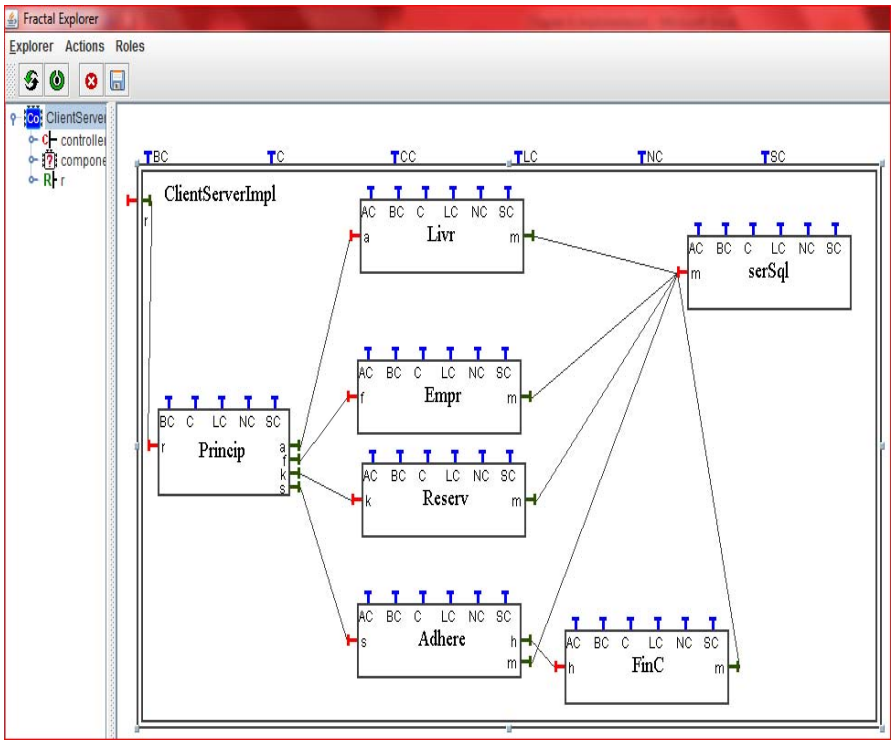


Figure 5.16 Architecture Fractal du système de gestion d'une bibliothèque

Dans notre démarche, nous avons créé :

- les composants primitifs

- une class Java qui va contenir la logique du composant (Adherent.java, Emprunt.java, Principale.java, Sql.java, FinCyc.java)

- des interfaces Java qui vont être associées avec les interfaces Fractal (AdherentINT.java, SqlINT.java, EmpruntINT.java,.....) chaque interface va exposer la méthode que nous voulons définir comme interface de composant (figure 5.17)

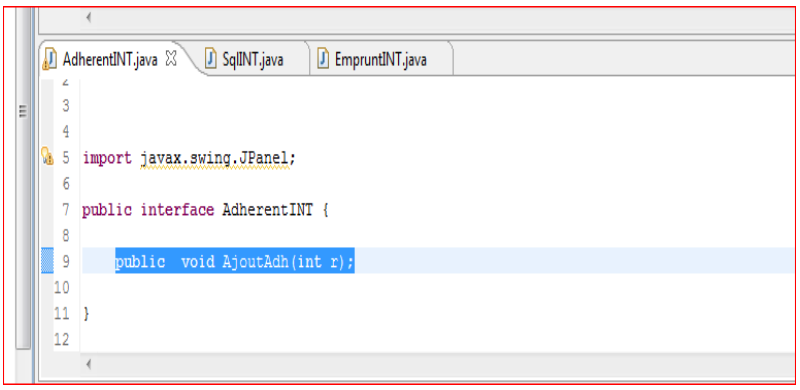


Figure 5.17 Interface Java du composant Adhérent Fractal

La figure 5.18 montre une partie de l'application générée.

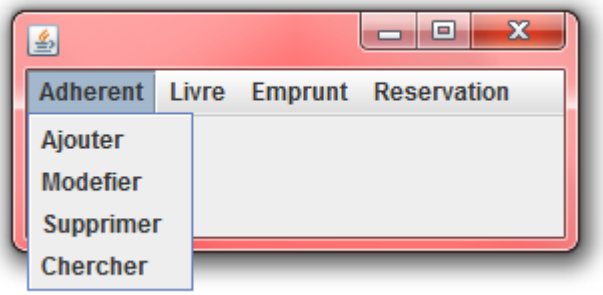


Figure 5.18 La fenêtre Principale

Conclusion

Dans ce chapitre, nous avons donné un aperçu sur la démarche proposée Onto-RUP à travers un exemple d'application simple. Nous avons donné les étapes nécessaires pour l'analyse des besoins d'évolution anticipée en utilisant les artéfacts élaborés durant les activités RE d'ONTO-RUP (cas d'utilisation et cas de changement). Dans l'approche RUP l'outillage du processus de développement est primordial pour assister les développeurs dans leurs activités tout au long du cycle de développement. Onto-RUP prévoit l'utilisation d'autres AGL (comme ORCTOOL) pour la maîtrise de l'analyse, la conception et l'implémentation des évolutions des besoins anticipées et non-anticipées.

CONCLUSION GENERALE

Les activités économiques utilisent de plus en plus de logiciels pour atteindre les objectifs souhaités. Ceci a conduit à des systèmes centrés logiciel de plus en plus complexes et soumis aux exigences croissantes des utilisateurs en termes de qualité et de fonctionnalités. Ces systèmes doivent en plus pouvoir évoluer au cours de leur existence afin de répondre de manière cohérente aux changements dans leur environnement, que ces derniers soient liés à l'évolution du marché, aux exigences des utilisateurs, ou à l'évolution des technologies. Ainsi, les systèmes logiciels doivent avoir une capacité d'adaptation importante.

De nos jours, faire évoluer un système logiciel est un défi réel. Pour cela de nombreux travaux en génie logiciel tentent d'apporter des solutions aux problèmes de l'évolution.

Bien que plusieurs approches soient actuellement proposées comme des solutions aux problèmes de l'évolution, elles demeurent néanmoins des palliatifs. De plus, les chercheurs admettent, actuellement, que les solutions aux problèmes de la maintenance nécessitent une approche interdisciplinaire qui traite toutes les facettes de l'évolution, étudie les contributions des approches et évalue leurs apports.

La prise en compte de l'ontogenèse constitue un nouveau défi et une vision radicale de l'évolution qui a un effet aussi bien sur notre façon de percevoir les systèmes logiciels que notre manière de les concevoir. Bien que l'approche ontogénétique apporte une vision radicale de l'évolution des systèmes qui imite le processus biologique de l'ontogénèse et est, de ce fait, censée éliminer de nombreux problèmes dont souffre les approches actuelles de l'évolution, nous avons très vite, constater que cette vision radicale pose des défis importants aux niveaux les plus abstraits du cycle de vie du logiciel. L'ontogenèse des systèmes logiciels fait référence à la capacité qu'a un logiciel à évoluer dynamiquement et d'une manière autonome qui lui permet de s'accommoder des exigences des utilisateurs et de l'évolution de ces dernières qu'elles soient anticipées ou non-anticipées. L'évolution d'un système ontogénétique a la particularité d'être un processus continu qui le transforme depuis sa création initiale.

Cette caractéristique n'est pas en accord avec les méthodes actuelles de développement qui considèrent que l'évolution est un processus sporadique. En conséquence, les plates-formes, les outils et les méthodologies que nous employons pour développer les systèmes logiciels ne sont pas appropriés et un effort considérable est nécessaire pour les adapter afin de supporter l'ontogénèse de logiciel. Pour tirer profit du paradigme ontogénétique, nous avons entamé dans ce travail de thèse une démarche de développement des systèmes ontogénétiques.

Pour cela, nous avons opté pour une extension d'une méthodologie existante de développement de systèmes logiciels. L'extension de méthodologie existante est une voie de recherche pragmatique. En effet, il n'est pas justifié dans un contexte universitaire d'investir dans la création complète d'une méthodologie dédiée. Deux catégories de méthodologies de développement de logiciels existent : les méthodes traditionnelles, qui sont axées-plan et sont lourdes, et celles dites agiles qui ont un caractère léger, flexible et adaptatif aux changements. Nous avons opté pour l'extension du Processus Unifié Rational (RUP), qui est l'une des méthodologies agile, qui se fixe comme objectifs d'assurer le développement de logiciels de haute qualité, qui respectent les besoins des utilisateurs, l'échéancier et le budget. L'extension concerne l'adjonction de nouvelles disciplines pour modéliser les évolutions anticipées et non anticipées et d'autres éléments de modélisation.

Concernant la discipline d'ingénierie des besoins des évolutions anticipées, l'approche ONTO-RUP propose un processus d'extraction de changements futurs ayant comme artéfacts d'entrée : le modèle orienté-but et le modèle de cas d'utilisation construit durant la discipline RE du RUP. Le processus d'extraction utilise la méthode « Roue des futurs ». Les artéfacts de sortie de cette discipline, étant le modèle des cas de changement.

Le travail entamé dans cette thèse part d'une nouvelle vision pour l'ingénierie des systèmes logiciels et il est nécessaire de rappeler qu'il présente un challenge pour plusieurs raisons :

- La complexité du processus RUP
- Le niveau d'abstraction élevé de l'ingénierie des besoins
- Le manque de travaux existants dans la littérature concernant les évolutions non anticipées
- Absence de travaux sur le développement continu de systèmes logiciels

Résumé des contributions

Nous estimons que ce travail nous a permis d'apporter certains points de vue, d'identifier des concepts, des mécanismes et d'en proposer d'autres. Nous avons maîtrisé le modèle Mage et son apport pour la modélisation de l'ontogenèse.

Nous avons fait une étude comparative entre les méthodologies de développement des systèmes logiciel traditionnelles et agiles, en présentant les points faibles et points forts de chacune d'entre elles.

Nous avons consacré une section importante dans le deuxième chapitre pour le formalisme UML, qui reste une référence incontournable dans le domaine du génie logiciel, permettant la représentation des modèles objets en intégrant différents diagrammes permettant de

structurer le système et son comportement. Néanmoins, le processus d'élaboration de ces modèles n'est pas décrit par UML. Le RUP etant basé sur UML et ce fait constitue une raison supplémentaire qui nous a fait opter pour l'extension de RUP plutôt que de proposer une nouvelle méthode. Le processus RUP a été présenté dans ce manuscrit avec plus de détails car il représente le noyau de la démarche proposée à différents niveaux : sa structure statique et dynamique, ses concepts, ses principes, etc.

L'extension du RUP représente la contribution la plus importante de ce travail de thèse. Il est nécessaire de noter que nous nous somme intéressés particulièrement à l'ingénierie des besoins d'évolution car c'est la phase la plus abstraite et critique dans le cycle de vie du système ontogénétique.

Nous avons proposé un processus pour l'ingénierie des besoins d'évolution en combinant plusieurs méthodes existantes dans la littérature (modèle des cas d'utilisation du RUP, modèle Futures wheel, le modèle orienté but i* des GORE) pour produire le modèle des cas de changement.

Perspectives de recherche

Les perspectives de ce travail sont nombreuses. Dans un premier lieu, il conviendra de combler l'approche d'ingénierie des besoins anticipés en proposant une approche de passage adéquate du modèle orienté but adapté vers le modèle de cas de changement.

Dans un second lieu, il est nécessaire d'envisager l'extension de RUP à différent niveaux et non plus seulement comme ajout de discipline d'ingénierie des besoins, préparation à la maintenance et la mise à jour dynamique.

Comme troisième perspective, il est question de considérer les changements non anticipés qui nécessitent la prise en compte de l'état actuel d'un système et les changements déjà existants dans son génotype. De même la vérification formelle des propriétés d'un système suite aux changements et le maintien de certaines qualités de service constituent une perspective qui nécessite beaucoup d'efforts. Aussi il est important de noter que l'aspect implémentation des changement et l'estimation des coûts des efforts sont des voies de recherches prometteuses.

REFERENCES

[Ada 98] C. Adami, Introduction to Artificial Life, Springer-Verlag, New York, Inc., 1998.

[Adi 13] F. Adikara, B. Sitohang, and B. Hendradjaya, "The Emergence of User Requirements Risk in Information System Development for Industry Needs," in *6th International Seminar on Industrial Engineering and Management*, 2013.

[Aiz 06] N. Aizenbud-Reshef, B. T. Nolan, J. Rubin, Y. Shaham-Gafni**,** Model traceability, *IBM Systems Journal*, Citeseer, 2006.

[Alm 06] Hyggo Almeida, Angelo Perkusich, Evandro Costa, Glauber Ferreira, Emerson Loureiro, Loreno Oliveira, and Rodrigo Paes. A Component Based Infrastructure to Develop Software Supporting Dynamic Unanticipated Evolution. In *Anais do XX Simposio Brasileiro de Engenharia de Software*, pages 145–160, Florian´opolis, SC, Brasil, 2006.

[Arm 05] Armarego, J., (2005),Studio Learning of Requirements: towards aligning teaching to practitioner needs*,In: 1st* International Workshop on Requirements Engineering Education and Training (REET) (2005).

[Arn 93] Arnold, R.S.: Software Reengineering. IEEE Computer Society Press (1993) [5, 291]

[Aur 05] A. Aurum and C. Wohlin, editors. Engineering and Managing Software Requirements. Springer Verlag, Berlin Germany, 2005.

[Bac 05] Bachmann, F. and Clements, P. C. (2005) 'Variability in Software Product Lines', Technical Report CMU/SEI-2005-TR-012, September 2005, Software Engineering Institute, Carnegie Mellon University.

[Ben 12] Lamia Ben Hiba, Mohammed Abdou Janati Idrissi, « Tendances des méthodes de gestion des projets », eTI, Revue électronique en Technologies de l'Information. http://www.revue-eti.net , 31 Mars 2012.

[Ben 13] Benabbas Zine Eddine, Un Outil Case pour la Manipulation des Cas de Changements, mémoire de fin d'étude Master, Département d'informatique, Université Badji Mokhtar-Annaba, 2013.

[Ber 98] Berry DM, Lawrence B (1998) Requirements engineering. IEEE Software 25(2): 26-29.

[Ber 07] Mario M. Ber´on, Pedro R. Henriques, Maria J. Varanda, Roberto Uzal, Program Inspection to inter-connect the Operational and Behavioral Views for Program Comprehension, First York Doctoral Symposium on Computing. York. England. 2007.

[Ber 98] Berry DM, Lawrence B ,Requirements engineering. IEEE Software 25(2): 26-29,(1998).

[Bie 03] G. Bierman et al, Formalising Dynamic Software Updating, Second International Workshop on Unanticipated Software Evolution, April 5 - 13, 2003, Warsaw, Poland, http://joint.org/use/2003/

[Bla 72] A.W. Blackman Jr., A mathematical model for trend forecasts, Technol. Forecast. Soc. Change 3 (1972)

441–452.

[Boe 81] B.Boem, “Software Engineering Economics”, Prentice Hall, 1981

[Boi 03] S. Boissau and J. C. Castella, “Constructing a common representation of local institutions and land-use systems through simulation gaming and multiagent modeling in rural areas of northern Vietnam: The sambaweek methodology,” Simulation & Gaming, vol. 34, pp. 342-357, 2003.

[Boo 91] G.Booch, Object Oriented Analysis and Design with Application, Benjamin/Cummings, 1991.

[Bor 13] Borjiba Othmane, Une conception des cas de changement basée composant, mémoire de fin d’étude Master, institut d’informatique, université Badji Mokhtar Annaba, 2013.

[Bub 94] Bubenko, J, A., Kirikowa, M., “Worlds” in Requirements Acquisition and Modeling, Sweden, 1994.

[Cas 92] X. Castellani, MCO, Méthodologie générale d’analyse et de conception des systèmes d’objets – l’ingénierie des besoins, Masson, 1992.

[Cas 02] Jaelson Castro, Manuel Kolp et John Mylopoulos. *Towards Requirements-Driven Information Systems Engineering : The Tropos Project* . Information Systems, vol. 27, pages 365–389, 2002.

[Cha 01] N. Chapin, J. E. Hale, K. Md. Khan, J.F. Ramil, and W. G. Tan. Types of software evolution and software maintenance. Journal of Software Maintenance and Evolution : Researche and Practice, 13:3-30, 2001.

[Chi 90] Chikofsky, E.J., Cross, J.H.: Reverse engineering and design recovery: A taxonomy. IEEE Software **7**(1) (1990) 13–17 [5, 106, 177, 303]

[Cis 14] Computer Information Systems, inc. http://www.cis.com

[Coc 01a] Cockburn A. and Highsmith J. (2001*),* Agile Software Development: The People Factor. *Computer*, Novembre 2001, 131-133.

[Coc 01b] A. Cockburn, Writing Effective Use Cases, Addison Wesley, New York, 2001.

[Col 10] Thomas COLLONVILLÉ, Elaboration de processus de développements logiciels spécifiques et orientés modèles, Thèse doctorat, l'Université de Haute-Alsace , 2010.

[Con 01] Constantine L., Methodological Agility, *Software Development*, 67-69, Juin 2001.

[Cop 98] J. Coplien et al., Commonality and variability in Software Engineering, IEEE Software, pp. 37-46, November/December 1998.

[Cys 03] Cysneiros, L.M., Kushniruk, A.: Bringing Usability to the Early Stages of Software Development. International Requirements Engineering Conf. IEEE(2003) 359-360

[Dar 93] Anne Dardenne, Axel van Lamsweerde et Stephen Fickas. Goal-directed requirements acquisition. Sci. Comput. Program., vol. 20, pages 3–50, April 1993.

[Dav 93] A.M. Davis, "Software requirements, Objects, functions and states", Prentice Hall, 1993.

[Del 96] F. Dellaert, R.D. Beer, A developmental model for the evolution of complete autonomous agents, In P. Maes, M. Mataric, J. Meyer, J. Pollack and S. Wilson (Eds.), From animals to animats 4: Proceedings of the Fourth International Conference on Simulation of Adaptive Behavior pp. 393-406, 1996.

[Dow 01] J. Dowling & V. Cahill, The K-Component architecture meta-model for self-adaptive software, Reflection 2001, The third international conference on metalevel architecture and separation of crosscutting concerns, Kyoto, Japan, September 2001.

[Eck 96] Ecklund, E. F., Delcambre, L. M. L. and Freiling, M. J. (1996) 'Change Cases: Uses Cases that Identify Future Requirements', Proceeding OOPSLA'96, ACM Press, New York.

[Enc 14a] Le système nerveux, Encyclopédie Microsoft Encarta en ligne, Microsoft Corporation, 2005, http://fr.encarta.msn.com

[Enc 14b] Le système immunitaire, Encyclopédie Microsoft Encarta en ligne, Microsoft Corporation, 2005, http://fr.encarta.msn.com

[Enc 14c] Le système endocrinien, Encyclopédie Microsoft Encarta en ligne, Microsoft Corporation, 2005, http://fr.encarta.msn.com

[Erd 03] Kagan Erdil,Emily Finn, Kevin Keating, Jay Meattle, Sunyoung Park, Deborah Yoon, Software Maintenance As Part of the Software Life Cycle, Comp180: Software Engineering Project, 2003.

[Est 12] Estier, T. and Métrailler, A. (2012) 'EVOLIS: a Framework for Evaluating Evolution of Information Systems', Actes du Colloque AIM 2012, Bordeaux, France, pp. 12.

[Eti 04] Etien, A., Rolland, C. and Salinesi C., (2004) 'Overview of a Gap-driven Evolution Process', AWRE'04,9th Australian Workshop on Requirement Engineering, Adelaide, Australia.

[Fis 98] Gerhard Fischer, Complex systems: Why do they need to evolve and how can evolution be supported, Lecture Notes in Computer Science Volume 1454, 1998, pp 92-112

[Fow 05] Fowler, M. (2005). The new methodology [agile methodology]. Software World, 36(1), 3-6.

[Ger 05] Germain, E., & Robillard, P. N. (2005). Engineering-based processes and agile methodologies for software development: a comparative case study. Journal of Systems and Software, 75(1-2), 17-27.

[Gil 03] S. F. Gilbert, Developmental biology, Seventh Edition, Sinauer associates Inc. Publishers, March 2003.

[Gir 07] Jean-Pierre Giraudin. «Complexité des systèmes d'information et de leur ingénierie». e-TI - la revue électronique des technologies d'information, Numéro 3, 9 mai 2007, http://www.revue-eti.netdocument.php?id=1180.

[Gle 72] Glenn, Jerome C., 1972. 'Futurizing Teaching vs Futures Course, Social Science Record, Syracuse University, Volume IX, No. 3 Spring 1972.

[Gle 09] Glenn, J. C. (2009) 'Futures Wheel', Futures Research Methodology Version 3.0, The Millennium Project, Washington, D.C.

[Gor 04] T. Gordon and J. Glenn, "Integration, Comparisons, and Frontiers of Futures Research Methods," New Technology Foresight, Forecasting and Assessment Methods, Seville, 2004.

[Gur 01] J. V. Gurp, J. Bosch, M. Svahnberg, On the Notion of Variability in Software Product Lines, Proceedings of WICSA 2001, August 2001.

[Hag 88] Hagelstein J., Declarative Approach to Information Systems Requirements, in Knowledge- Based Systems, Vol No4, 1988.

[Hai 08] Hainaut, J. L., Anthony, C., Henrard, J. and Hick, J. M. (2008) 'Migration of Legacy Information Systems', in Mens, T. and Demeyer, S. (eds.), 'Software Evolution'. DOI 10.1007/978-3-540-76440-3, Springer.

[Heu 94] J. C. Heudin, La vie Artificielle, Edition Hermes, 1994.

[Hic 01] M. Hicks. Dynamic Software Updating. PhD thesis, Department of Computer and Information Science, University of Pennsylvania, August 2001.

[Hum 88] HUMPHREY, W. S. (1988). The software engineering process: definition and scope. In Proceedings of the 4th international software process workshop on Representing and enacting the software process, pages 82–83, New York, NY, USA. ACM Press.

[IBM 12] IBM corporation, Rational RequisitePro, http://www-306.ibm.com/software/awdtools/reqpro/, visited 20/05/2014.

[IBM 15] http://www.ibm.com/developerworks/rational/library/jun07/cuellar/index.html, visited 11/01/2015

[Jac 01] Michael Jackson. Problem frames : analyzing and structuring software development problems. Addison-Wesley Longman Publishing Co.,Inc., Boston, MA, USA, 2001.

[Joh 05] Bruce Johson, W W.Woolfolk, R. Miller, “Flexible Software Design”, Auerbach Publications, 2005.

[Kav 06] Kavakli, E. and Loucopoulos, P, Experiences with Goal-Oriented Modelling of Organisational Change, IEEE Transactions on Systems, Man and Cybernetics-Part C, Vol N36, Issue 2, pp.221-235, 2006.

[ket 05] Kettunen, P., & Laanti, M., How to steer an embedded software project: tactics for selecting the software process model. *Information and Software Technology*, *47*(9), 587-608, 2005.

[Khe 9a] Kherissi, F. and Meslati, D.,Developing Ontogenetic Software Systems,International Symposium on Programing Systems ISPS, Algiers, 2009.

[Khe 9b] Kherissi, F. and Meslati, D.,ONTO-RUP: A RUP Based Approach for Developing Ontogenetic Software Systems, Lecture Notes of the Institute for Computer Sciences, Social Informatics and Telecommunications Engineering, volume 39, pp 140-151, Springer, 2009, ISSN: 1867-8211.

[Khe 14] Kherissi, F. and Meslati, D., and Tamzalit, D., An approach based on extending the RUP for dealing with anticipated changes in ontogenetic Software Systems, To appear in IJCAT (Internationa Journal for Compter Applications in technology), Inderscience Publisher, Vol. 51, No. 4.

[Kis 03] T. Kistler, M. Franz, Continuous program optimization : A case stydy, ACM Transactions on Programming Languages and Systems, Vol. 25, No. 4, pp 500-548, July 2003.

[Kni 02] Gunter Kniesel, Joost Noppen, Tom Mens, and Jim Buckley, Unanticipated Software Evolution, *The First International Workshop on Unanticipated Software Evolution.* 10-14 June 2002, Malaga, Spain.

[Kot 98] G. Kotonya and I. Sommerville, Requirements Engineering: Processes and Techniques, John Wiley & Sons, 1998.

[Kro 03] P. Kroll, and P. Kruchten, The Rational Unified Process Made Easy: A Practitioner's Guide to the RUP, Addison Wesley, 2003.

[Kru 00] Philippe Kruchten. The Rational Unified Process An Introduction, Second Edition. Addison Wesley, 2000.

[Kru 01] P. Kruchten, "Software Maintenance Cycles with the RUP", The Rational Edge, August 2001.

[Lam 01] A. V. Lamsweerde, Goal-oriented requirements engineering: A guided tour, in Proceedings of the 5th IEEE International Symposium on Requirements Engineering, Toronto, Canada, pp. 249-262, 2001.

[Lam 02] Axel van Lamsweerde et Emmanuel Letier, From Object Orientation to Goal Orientation : A Paradigm Shift for Requirements Engineering, in Martin Wirsing, Alexander Knapp et Simonetta Balsamo, editeurs, RISSEF, volume 2941 of Lecture Notes in Computer Science, pages 325–340. Springer, 2002.

[Lam 09] Axel van Lamsweerde , Requirements Engineering : From System Goals to UML Models to Software Specifications, Wiley,2009.

[Lar 03] Larman, C., Agile and Iterative Development: A Manager's Guide, Addison-Wesley, 2003.

[Lew 99] B. Lewin, Genes VII, Oxford University Press, Oxford, 1999.

[Li 09] Tong Li , An Approach to Modeling Software Evolution Processes, ISBN: 978-3-540-79463-9, Springer Berlin Heidelberg New York, 2009.

[Lin 91] F. Lints, Génétique, Troisième édition, Lavoisier Technique et Documentation, 1991.

[Liu 01] Lin Liu et Eric Yu, From Requirements to Architectural Design : Using Goals and Scenarios. In First International Workshop From Software Requirements to Architectures (STRAW 01), Toronto, Canada, May 2001.

[Lod 04] Lodding, K. N., Hitchhiker’s Guide to Biomorphic Software, QUEUE, Vol. 2, N°4,pp.66-75, ACM, June 2004.

[Lov 96] D. Loveridge, Technology Foresight and Models of the Future, Policy Research in Engineering, Science and Technology, September 1996.

[Mao 07] Y. Mao, J. Vassileva, and W. Grassmann, A System Dynamics Approach to tudy Virtual Communities, in Proceedings of the 40th Annual Hawaii International Conference on System Sciences (HICSS), January 2007.

[McD 84] John McDermid et Knut Ripken, Life cycle support in the ADA environment, University Press, 1984.

[Mea 05] Nancy R. Mead, Ted Stehney. "Security Quality Requirements Engineering (SQUARE) Methodology." SESS 2005, 2005.

[Men 08] T.Mens, Serge Demeyer, Software Evolution, Springer Verlag, 2008.

[Mes 04] D. Meslati & al, The MAGE Ontogenetic Model: Towards autonomously-developed software, 17th Int. Conf. on Software & Systems Engineering and their Applications, Paris, November 2004.

[Mes 06] D. Meslati, Mage: Une approche ontogénétique de l'évolution dans les systèmes logiciels critiques et embarqués, Thèse de Doctorat d'Etat, Université de Annaba, Février 2006.

[Mes 08] Véronique Messager Rota, Gestion de projet, Vers les méthodes agiles, Eyrolles, édition 2008.

[Moi 90] J.L.LeMoigne, La modélisation des systèmes complexes, Dunod, Paris, 1990.

[Mul 00] P. A. Muller, and N. Gaertner "Modélisation objet avec UML", Eyrolles, 2000.

[Mur 04] Murch, R. , Autonomic Computing On Demand Series, Prentice Hall, March 2004.

[Naw 02] Nawrocki, J. R., Walter, B., & Wojciechowski, A., Comparison of CMM Level 2 and extreme programming, 7th European Conference on Software Quality (ECSQ 2002), (pp. 288-297). Helsinki, Finland: Springer-Verlag, 2002.

[Nel 05] R. J. Nelson, An introduction to behavioral endocrinology, Third edition, Sinauer associates Inc. Publishers, 2005.

[Ner 05] Nerur, S., Mahapatra, R., & Mangalaraj, G., Challenges of migrating to agile methodologies. Communications of the ACM, 48(5), 72-78, 2005.

[Nur 02] Selmin Nurcan, Judith Barrios, Colette Rolland, Une méthode pour la définition de l'impact organisationnel du changement, INFORSID 2002

[Nus 00] Bashar Nuseibeh et Steve Easterbrook, *Requirements engineering : a roadmap, i*n Proceedings of the Conference on The Future of Software Engineering, ICSE '00, pages 35–46. ACM, 2000.

[OMG 09] Object Management Group (OMG), 2009. Spécification (UML) Unified Modeling Language., http://www.omg.org/spec/UML/2.2/.

[Ous 99] C. Oussalah, D. Tamzalit, L'émergence comme processus d'évolution d'objet In Ed. C. Oussalah, Génie Objet : Analyse et Conception de l'Evolution, Edition Hermes, 1999

[O'B 03] Micheal P. O'Brien. Sofware Comprehension - A Review and Research Direction. Technical Report, 2003

[Pas 06] Passing, J.,Requirements Engineering in the Rational Unified Process, Seminar Requirements Engineering, Summer term, Hasso Plattner Insitute for Software Systems Engineering, 2006. http://int3.de/res/RUP/RUP_Paper_JohannesPassing.pdf.

[PCD 14] PDCA, la roue de Deming, http://www.logistiqueconseil.org/Articles/Methodes-optimisation/Pdca-roue-deming.htm, visité Novembre 2014.

[Per 07] J-M. Perronne. Une contribution Objet pour la conception de systèmes logiciels de commande plus sûrs. Habilitation à diriger des recherches. Université de Haute-Alasace, 2007.

[Phi 00] Phillips, D., The Software Project Manager's Handbook - Principles that work at Work. 1st ed. Los Alamitos, California, IEEE Computer Society, 2000.

[Pim 11] Pimentel, J., Castro, J., Perrelli, H., Emanuel, S., and Franch, X., Towards Anticipating Requirements Changes through studies of the future, In Proceedings of The 5th IEEE International Conference on Research Challenges in Information Science, May 19-21, Gosier, Guadeloupe, (2011).

[Ras 08] Awais Rashid et Ruzanna Chitchyan. *Aspect-oriented requirements engineering : a roadmap*. In Proceedings of the 13th international workshop on Early Aspects, EA '08, pages 35–41, New York, NY, USA, 2008. ACM.

[Rid 96] M. Ridley, Evolution, Second edition, Blackwell scientific publications Ltd, Oxford, 1996.

[Rol 98] Colette Rolland, Carine Souveyet et Camille Ben Achour, *Guiding Goal Modeling Using Scenarios*. IEEE Trans. Softw. Eng., vol. 24, pages 1055–1071, December 1998.

[Rol 03] Colette Rolland, "L'INGENIERIE DES BESOINS : L'APPROCHE L'ECRITOIRE", Journal Techniques de l'Ingénieur (2003) 1.

[Rol 05] Rolland C., L'ingénierie des méthodes : une visite guidée , *e-TI*, n° 1, (http://www.revueeti.net/document.php?id=726), 25 octobre 2005.

[Roy 70] Royce, W.W., Managing the development of large software systems, IEEE WESCON, 1970.

[Rum 96] J. Rumbaugh, A State of Mind : Modeling behavior, Modeling & Design, JOOP, Volume 9, Number 4, pp. 6-12, July/August 1996.

[Run 04] Per Runeson Peter Greberg Extreme Programming and Rational Unified Process - Contrasts or Synonyms? SERPS´04 - Software Engineering Research and Practice in Sweden, 2004

[Sac 10] Sachidanandam Sakthive , Manage Requirements Volatility to manage Risks in IS development projects, ISACA Journal, Volume N°5, 2010.

[Sad 03] Salah Sadou, Rapport d'HDR de l'université de Bretagne Sud 2003. http://www-irisa.univ-ubs.fr/Salah.Sadou/papers/hdr.pdf

[Sal 04] Salinesi, C., Etien, A., and Wäyrynen, J., Towards a Systematic Propagation of Evolution Requirements in IS Adaptation Projects, Australian Conference on Information Systems (ACIS), Hobart, Australia, 2004.

[San 02] Santander, V. F. A. and Castro, J. F. B.,Integrating Use Cases and Organizational Modeling, SBES02 - XVI Brazilian Symposium on Software Engineering, Gramado. October, 2002.

[Sha 67] J.L. Shaerer and H.H. Richardson. Introduction to System Dynamics. Addison Wesley, NewYork, 1967.

[Shu 03] U. P. Shultz et al., Automatic Program Specialization for Java, ACM Transactions on Programming Languages and Systems, Vol. 25, No. 4, pp 452-499, July 2003.

[Som 01] I., Sommerville, Software engineering, 6th ed. Harlow, Pearson Education Ltd, 2001.

[Som 06] I. Sommerville. *Software Engineering 8th Revised edition*. Addison-Wesley Educational Publishers Inc, United Kingdom, 2006.

[Sri 94] M. Srinivas, L. M. Patnaik, Genetic Algorithms: A survey, Computer, pp. 17-26, June 1994.

[Sut 03] Alistair Sutcliffe. Scenario-Based Requirements Engineering. In Proceedings of the 11th IEEE International Conference on Requirements Engineering, Washington, DC, USA, 2003. IEEE Computer Society.

[Tem 02] G. Tempesti et al., A POEtic Architecture for Bio-Inspired Hardware, 8th Int. Conf. on the Simulation and Synthesis of Living Systems, Sydney, Australia, Dec. 2002, MIT Press,Cambridge, 2002, pp 111-115.

[Tes 00] G. J. Tesauro and J. O. Kephart, "Foresight-based pricing algorithms inagent economies," Decision Support Systems, vol. 28, issues 1-2, pp.49-60, March 2000.

[Tre 07] R.Tremblay, Implantation d'une méthode Agile de développement Logiciel en entreprise : Une culture accueillant le changement, mémoire pour l'obtention du grade de maître (M. Sc.), université de Laval, 2007.

[Uni 14a] Encyclopædia Universalis en ligne, Chapitre Evolution, Mars 2004 www.universalis.edu.com/private/

[Wau 13a] Yves Wautelet and Manuel Kolp. Mapping i* within uml for business modeling. In Joerg Doerr and Andreas L. Opdahl, editors, REFSQ, volume 7830 of Lecture Notes in Computer Science, pages 237-252. Springer, 2013.

[Wau 13b] Yves Wautelet, and Manuel Kolp, On the Integration of i* into RUP, Proceedings of the 6th International i* Workshop (iStar 2013), CEUR Vol-978

[Yu 95] Eric Yu. Modelling Strategic Relationships for Process Reengineering. Ph.D. Thesis, Department of Computer Science, University of Toronto, Canada, 1995.

[Yu 97] Eric Yu. *Towards Modeling and Reasoning Support for Early-Phase Requirements Engineering*. In Proceedings of the 3rd IEEE International Symposium on Requirements Engineering, RE '97, pages 226–235, Washington, DC, USA, 1997. IEEE Computer Society.

[Yu 12] Yu, B. L., Wooi, K. L., Wai, Y. T. and Soo, F. T. (2012) 'Software Development Life Cycle AGILE vs Traditional Approaches', International Conference on Information and Network Technology, IPCSIT, vol. 37, IACSIT Press, Singapore.

[Yue 87] K. Yue. What Does ItMean to Say that a Specification is Complete ? In Proceeging of the IWSSD-4, Fourth International Workshop on Software Specification and Design, 1987.

[Zav 93] Z AVE ,P. AND J ACKSON , M. 1993. Conjunction as composition. ACM Trans. Softw. Eng. Methodol. 2 , 4 (Oct.), 379–411.

[Zet 01] Zettel, J., Maurer, F., Munch, J., & Wong, L., LIPE: a lightweight process for e-business startup companies based on extreme programming, Third International Conference on Product Focused Software Process Improvement (PROFES 2001), (pp. 255-270). Kaiserslautern, Germany: Springer-Verlag., 2001

[Zow 96] D. Zowghi, A.K. hose and P. Peppas, A Framework for Reasoning about Requirements Evolution, Proceedings of PRICAI96, Australia, 1996.

Résumé

L’ontogenèse des systèmes logiciels se réfère à la capacité des logiciels à évoluer dynamiquement et de façon autonome pour répondre aux besoins des utilisateurs et leurs changements anticipés et non anticipés. L'évolution d'un système ontogénétique a la particularité d'être un processus continu qui le façonne depuis le début de sa création. Cette particularité ne correspond pas aux méthodes actuelles de développement qui considèrent que l'évolution est un processus sporadique. Par conséquent, les plateformes, outils et méthodes que nous utilisons pour développer des systèmes logiciels ne sont pas adaptés à la modélisation de l'ontogenèse du logiciel. L’approche proposée dans cette thèse, nommée Onto-RUP, présente une démarche pour le développement des systèmes ontogénétiques basée sur une extension du Rational Unified Process (RUP). Cette extension porte sur l’adjonction de deux nouvelle disciplines et des artéfacts spécifiques.

Mots clés. Evolution, RUP, développement logiciel, système ontogénétique

ملخص

إن تطور أنظمة البرمجيات يشير إلى قدرة البرامج للارتقاء بشكل ديناميكي و ذاتي لتلبية احتياجات المستخدمين والاحتياجات المتوقعة وغير المتوقعة التغيير. إن ارتقاء النظام التطوري لديه خصوصية هي انه عملية مستمرة تشكله منذ بداية إنشائه. هذه الخصوصية لا تتناسب مع أساليب التطوير الحالية التي تعتبر الارتقاء عملية متفرقة في الزمن. وبالتالي الأسس والأدوات والأساليب التي نستخدمها لتطوير أنظمة البرمجيات لا تتناسب لقيادة تطور البرمجية. و تعرض المقاربة "أنتوروب" المقترحة في هذه الأطروحة، نهجا لتطوير الأنظمة التطورية على أساس امتدادا لروب. يشمل هذا الامتداد إضافة اختصاصين جديدين وبعض عناصر النمذجة.

كلمات مرشدة. إرتقاء، روب، تطوير البرمجيات، نظام تطوري