ÉCOLE DE TECHNOLOGIE SUPÉRIEURE
UNIVERSITÉ DU QUÉBEC

THÈSE PRÉSENTÉE À
L'ÉCOLE DE TECHNOLOGIE SUPÉRIEURE

COMME EXIGENCE PARTIELLE
À L'OBTENTION DU
DOCTORAT EN GÉNIE
Ph.D.

PAR
Samir KHERRAF

MÉTHODOLOGIE DE TRANSFORMATION DU CIM EN PIM DANS L'APPROCHE MDA

MONTRÉAL, LE 20 DÉCEMBRE 2011

CETTE THÈSE A ÉTÉ EVALUÉE

PAR UN JURY COMPOSÉ DE :

M. Witold Suryn, directeur de thèse
Département du génie logiciel et des technologies de l'information à l'École de technologie supérieure

M. Alain Abran, co-directeur de thèse
Département du génie logiciel et des technologies de l'information à l'École de technologie supérieure

M., Pierre Bourque, membre du jury
Département du génie logiciel et des technologies de l'information à l'École de technologie supérieure

M., Tony Wong, président du jury
Département de génie de la production automatisée à l'École de technologie supérieure

M., Chadi El Zammar, examinateur externe
Ericsson Canada

ELLE A FAIT L'OBJET D'UNE SOUTENANCE DEVANT JURY ET PUBLIC

LE 24 NOVEMBRE 2011

A L'ÉCOLE DE TECHNOLOGIE SUPÉRIEURE

À la mémoire du professeur Éric Lefebvre

REMERCIEMENTS

Je dédie cette thèse à mon codirecteur M. Éric Lefebvre, qui nous a quittés sans la voir achevée. Éric aura été d'un total soutien tout au long de cette aventure, aussi bien professionnellement – son suivi sans faille, sa rigueur, sa ponctualité – qu'humainement – son perpétuel sourire, sa gentillesse et sa chaleur. Sa retraite puis sa maladie ne l'auront jamais distrait de son rôle de mentor pour moi comme pour mon travail. *In memoriam*.

Je remercie ensuite M. Alain Abran qui a très généreusement et au pied levé accepté de remplacer Éric à la codirection de ma thèse. Là encore, rigueur, pragmatisme, sens du détail, conseils et commentaires judicieux auront grandement facilité la dernière année de mes travaux. En outre, Alain m'a inculqué une toute autre façon d'aborder la recherche scientifique.

M. Witold Suryn, mon directeur de thèse, a également grandement contribué à orienter mon travail par ses questions pertinentes et précises qui m'ont souvent amené à remettre en question mes certitudes. Parallèlement, il a su m'insuffler beaucoup d'énergie en me rappelant régulièrement la confiance qu'il plaçait dans mes capacités et l'issue de mes travaux.

Les membres de mon jury, MM. Pierre Bourque, Tony Wong et Chadi El Zammar, pour leur commentaires instructifs à l'égard de cette thèse et bien plus.

Alexandre Moïse, mon excellent collègue et ami, a toujours su prendre le temps de me faire part de sa vision, de ses critiques sur mon projet de recherche.

Son père Jean-Claude m'a fait suffisamment confiance pour m'embarquer dans l'aventure de la création d'entreprise, et pour faire de moi un élément important d'Almonix. En outre, il m'a permis d'effectuer une partie de mes travaux de recherche dans cette compagnie, en collaboration avec le MITACS.

Jonathan Gareau, en acceptant de poursuivre la voie de mes recherches, pérennise les résultats de mes travaux. Son projet de maîtrise porte sur la validation empirique de certains concepts de ma thèse, ce qui s'annonce un choix courageux !

Antoine Palangié, mon compagnon de galère, de voyages et d'aventure qui a accepté de m'accompagner en Australie pour présenter une partie de mes recherches aux antipodes. Il aura aussi relu ma thèse sans en comprendre le moindre détail, selon son propre aveu. Il aura été disponible pour me remonter le moral dans mes phases de doute et de découragement depuis le tout début de mon doctorat.

Ma famille et mes parents, Rabia (mon éducatrice), Bouchaïb (mon prof), Karima (ma financière), Rajaâ (ma banquière), Khalid (mon agronome-journaliste) et Amine (mon médecin), ont fait de moi ce que je suis.

Caroline Bergeron, la femme qui m'aime et partage ma vie, m'aura soutenu lors d'innombrables soirées devant l'ordinateur. Elle aura finalement été ma compagne dans cet épique voyage de 24 000 km en voiture jusqu'à l'océan Arctique que nous nous sommes offerts à la fin de ma rédaction.

La famille de Caroline, Jacinthe, Conrad, Mélanie et Mathieu, en étant constamment derrière elle, aura aussi toujours été derrière moi.

MÉTHODOLOGIE DE TRANSFORMATION DU CIM EN PIM DANS L'APPROCHE MDA

Samir KHERRAF

RÉSUMÉ

L'*Object Management Group* (OMG) a proposé une nouvelle approche de développement de logiciel nommée Model Driven Architecture (MDA). Cette approche met l'accent sur l'élaboration des modèles de plus haut niveau d'abstraction et favorise l'approche de transformation d'un modèle à l'autre. MDA préconise l'élaboration des trois types de modèles suivants :

- *Computation Independent Model* (CIM) : ce modèle représente le plus haut niveau d'abstraction et décrit les exigences du système ainsi que sa manière de fonctionner dans son environnement tandis que les détails de la structure de l'application et de la réalisation sont cachés ou encore indéterminés.
- *Platform Independent Model* (PIM) : ce modèle décrit les détails du système sans montrer les détails spécifiques à une plateforme d'exécution ou à une technologie particulière.
- *Platform Specific Model* (PSM) : ce modèle décrit les détails et les caractéristiques supprimés du PIM. Il doit être adapté pour spécifier l'implémentation du système dans une seule et unique plateforme technologique.

Comme ces différents types de modèles représentent différents niveaux d'abstraction d'un même système, MDA recommande l'utilisation de mécanismes de transformation permettant les transformations du CIM vers le PIM et du PIM vers le PSM.

Depuis l'avènement de MDA, plusieurs travaux de recherche ont abordé la problématique de transformation du PIM vers le PSM et du PSM vers le code mais très peu traitent de la transformation du CIM vers le PIM. Bien que la littérature présente quelques travaux reliés à cette question, il semble que peu de chercheurs se soient penchés sur les problèmes reliés à la transformation du CIM vers le PIM. Ainsi, le CIM a été initialement considéré comme partie intégrante du PIM. Bien que la notion de l'indépendance de la plateforme soit assez claire, la notion du concept ''Computation'' reste floue. Par conséquent, la frontière entre les modèles CIM et PIM reste aussi vague.

Dans le but de transformer le CIM en PIM, nous avons identifié les trois problématiques de recherche suivantes : 1) la définition de l'architecture du CIM permettant de circonscrire ses frontières par rapport au PIM, 2) la définition de l'architecture du PIM permettant de circonscrire ses frontières par rapport au PSM, 3) la définition d'une méthodologie permettant de transformer le CIM en PIM.

La contribution de cette thèse s'inscrit dans le domaine de l'ingénierie dirigée par les modèles. Nous y proposons : 1) une architecture du CIM basée sur la composition de trois modèles *Business Motivation Model* (BMM), *Business Process Model* (BPM) et *Requirement*

Model (RM), 2) une architecture du PIM basée sur les patrons d'analyse et les patrons archétype, 3) une méthodologie couvrant l'ensemble des étapes de création du CIM ainsi que les techniques et les artefacts à produire, permettant la transformation du CIM en PIM. Ce travail contribue de plus à l'amélioration de la traçabilité entre le CIM et le PIM ainsi qu'à la réduction du fossé entre les activités des analystes d'affaires et des architectes de logiciels.

Mots-clés : Model Driven Engineering (MDE), Model Driven Architecture (MDA), Computation Independent Model (CIM), Platform Independent Model (PIM), Business Process Model (BPM), Requirement Model (RM), Business Motivation Model (BMM), Archetypes Pattern, Analysis Pattern, Domain Neural Component (DNC), Business Component, Transformation Rules.

MÉTHODOLOGIE DE TRANSFORMATION DU CIM EN PIM DANS L'APPROCHE MDA

Samir KHERRAF

ABSTRACT

The Object Management Group (OMG) has proposed a new software development approach called Model Driven Architecture (MDA). This approach focuses on developing models of the highest level of abstraction and supports the transformation approach from one model to another. MDA advocates the development of three types of models:

Computation Independent Model (CIM): this model represents the highest level of abstraction. It describes the system requirements and the environment in which it will operate while the details of the software structure and realisation are hidden or not yet determined.

Platform Independent Model (PIM): this model describes the details of the system, but does not show details of the use of its platform or of a particular technology.

Platform Specific Model (PSM): this model describes the details and features absent from the PIM. It must be adapted to specify the implementation of the system in a single technology platform.

As these different types of models represent various levels of abstraction of the same system, MDA recommends the use of transformation mechanisms allowing the transformation of the CIM to PIM and the PIM to PSM.

Since the advent of MDA, many research studies have addressed the issue of transforming the PIM to PSM and PSM to code, but very few deal with the transformation of the CIM to PIM.

Although the literature presents some work related to this issue, it seems that few researchers have studied the problems related to the transformation of the CIM to PIM. The CIM was originally considered an integral part of the PIM. Although the concept of independence of the platform is quite clear, the notion of the concept Computation remains unclear. Therefore, the boundary between CIM and PIM models also remains vague.

In order to transform the CIM to PIM, we have identified three research issues: 1) the definition of the architecture of the CIM to define its borders with the PIM, 2) the definition of the architecture of the PIM to define its borders with the PSM, 3) the definition of a methodology to transform the CIM to PIM.

This thesis contributes to the field of model driven engineering. More specifically we propose for this end: 1) an architecture of the CIM based on the composition of three models

Business Motivation Model (BMM), Business Process Model (BPM) and Requirement Model (RM), 2) an architecture of the PIM based on analysis patterns and archetype patterns, 3) a methodology covering all creation steps of the CIM and the techniques and artefacts to produce it or them, allowing the transformation of the CIM to the PIM. A contribution of this work is to improve the traceability between CIM and PIM, as well as bridging the gap between the activities of the business analysts and of the software architects.

Keywords: Model Driven Engineering (MDE), Model Driven Architecture (MDA), Computation Independent Model (CIM), Platform Independent Model (PIM), Business Process Model (BPM), Requirement Model (RM), Business Motivation Model (BMM), Archetypes Pattern, Analysis Pattern, Domain Neural Component (DNC), Business Component, Transformation Rules.

TABLE DES MATIÈRES

Page

LISTE DES TABLEAUX

Page

LISTE DES FIGURES

Page

LISTE DES ABRÉVIATIONS, SIGLES ET ACRONYMES

4GL	Fourth-Generation Programming Language
ACP	Actor Communication Pattern
ATL	Atlas Transformation Language
BC	Business Component
BCE	Boundary Control Entity
BCM	Business Component Model
BEC	Business Entity Component
BIAIT	Business Information Analysis and Integration Technique
BICS	Business Information Charecterization Study
BMM	Business Motivation Model
BPC	Business Process Component
BPM	Business Process Model
BPMN	Business Process Model and Notation
BSP	Business Service Planification
CASE	Computer Aided Software Engineering
CIM	Computation Indepenedent Model
CORBA	Common Object Request Broker Architecture
CRUD	Create Read Update Delete
DNC	Domain Neutral Component
EAI	Enterprise Application Integration
EBP	Elementary Business Process

EJB	Enterprise Java Beans
GRASP	General Resposibility Assignment Software Patterns
IBM	International Business Machines
IDM	Ingenièrie Dirigée par les Modèles
JEE	Java Enterprise Edition
MDA	Model Driven Architecture
MDE	Model Driven Engineering
MI	Moment-Interval
MOF	Meta Object Facility
OCL	Object Constraint Language
ODP	Open Distributed Processing
OMG	Object Management Group
OMT	Object Modeling Technique
OOSE	Object-Oriented Software Engineering
PIM	Platform Independent Model
PPT	Party Place Thing
PSM	Platfrom Independent Model
QVT	Query Transformation Language
RM	Requirement Model
RUP	Rational Unified Process
SCP	System Communication Pattern
SOA	Software Oriented Architecture
SoaML	Service Oriented Architecture Modeling Language

TFMMDA	Topological Functioning Modeling for Model Driven Architectur
UCM	Use Case Model
UML	Unified Modeling Language

INTRODUCTION

La modélisation est une activité principale dans tous les domaines de l'ingénierie. Elle est primordiale lors de l'analyse et de la conception des systèmes complexes. Les modèles sont des abstractions du système et de ses environnements. Ils permettent de répondre aux préoccupations des ingénieurs par la réalisation d'un prototype ou la modification dans une conception. Chaque modèle, créé pour une finalité, est généralement moins coûteux à produire qu'un système réel.

L'application de la modélisation dans le développement logiciel a commencé entre le milieu des années 1970 et la fin des années 1980. Durant la période comprise entre l'année 1989 et l'année 1994, l'industrie du logiciel a vu émerger de nombreuses méthodes de conception et d'analyse, chacune avec sa propre approche de modélisation et de notation. À la fin de 1994, Grady Booch, Jim Rumbaugh et Ivar Jacobson ont uni leurs efforts afin de créer un langage de modélisation unifié tout en se basant respectivement sur leur propre méthodologie de développement logiciel : la méthode Booch, OMT (Object Modeling Technique) et OOSE (Object-Oriented Software Engineering). Les efforts de Booch, Rumbaugh et Jacobson ont abouti à la proposition du langage UML (Unified Modelling Language) qui fut adopté par l'OMG (Object Management Group) en 1997 comme standard de modélisation (Martin Fowler et Kendall Scott, 1999). UML s'est imposé depuis sa standardisation comme la spécification de l'OMG la plus utilisée dans le domaine de l'ingénierie du logiciel (OMG, 2011).

Pourtant, l'utilisation des modèles se réduit, dans la plupart des cas, à des fins de documentation, car la relation entre les modèles et leurs implémentations est seulement intentionnelle et non formelle. Ceci pose deux problématiques. D'une part, les systèmes logiciels ne sont pas stables lors du processus de développement et sont susceptibles à des changements significatifs, particulièrement durant la première phase de leur cycle de développement (la phase des exigences). Dans ce cas-ci, la documentation doit être adaptée

méticuleusement, ce qui peut devenir une tâche complexe. D'autre part, la documentation des modèles n'évolue pas lorsque ce sont les programmeurs du code source qui prennent la relève dans le processus de développement : les modèles deviennent alors obsolètes et moins utiles avec le temps.

L'ingénierie dirigée par les modèles est arrivée avec une approche complètement différente. Les modèles ne sont plus contemplatifs lorsque l'implantation dans du code source est automatisée : les modèles sont alors considérés à un niveau égal au code source (Joaquin Miller et Jishnu Mukerji, 2003).

L'approche MDA (Model Driven Architecture) (Joaquin Miller et Jishnu Mukerji, 2003), une variante de l'ingénierie des modèles préconisée par l'OMG (Object Management Group), vise à travers les modèles CIM, PIM et PSM, à trouver des abstractions spécifiques au domaine d'affaires et à les rendre accessibles via la modélisation formelle. Ce processus crée un potentiel pour automatiser la production de logiciels, ce qui permet d'en augmenter la productivité, la qualité et la maintenabilité (Jean Bézivin et al., 2005; Jean Bézivin et Olivier Gerbé, 2001). En fait, l'adjectif ''Driven'' de MDA souligne que ce paradigme attribue aux modèles un rôle central et actif qui les rend aussi importants que le code source.

C'est dans cette lignée de l'ingénierie des modèles que s'inscrivent les travaux de cette thèse. Plus précisément, l'accent sera mis sur la définition de l'architecture du CIM et de l'architecture du PIM tout en identifiant l'ensemble des éléments qui les constituent ainsi que les relations qui les interrelient, en plus d'une méthodologie permettant la transformation du CIM en PIM.

Structure de la thèse

Cette thèse porte sur la création et l'expérimentation d'une méthodologie qui permette la transformation du CIM vers le PIM dans l'approche MDA. Ainsi, le chapitre 1 consiste en une revue de la littérature sur l'ingénierie des modèles : il s'agit principalement de mettre en

contexte la problématique de la transformation du CIM vers le PIM dans l'approche MDA. De plus, l'importance de la modélisation et de la métamodélisation est expliquée.

Le chapitre 2 présente l'objectif principal, les sous-objectifs et la méthodologie de recherche.

Le chapitre 3 présente l'architecture du CIM. Celui-ci, est construit à partir de la composition de trois modèles le BMM, le BPM et l'UCM. Les éléments de ces derniers ainsi que les relation entre ceux-ci sont présentées.

Le chapitre 4 présente l'architecture du PIM. Celui-ci, s'appuie sur les quatre archétypes de Coad (Moment-Interval, Role, Party-Place-Thing et Description) et le Domain Neutral Component (DNC). De plus, des patrons d'exigences et des patrons d'analyses, nécessaires pour aligner le PIM avec le CIM, sont expliqués.

Le chapitre 5 présente la méthodologie de la conception du modèle des processus d'affaires (BPM). Celui-ci, fait partie intégrante du CIM. Un métamodèle du BPM est construit à cette fin. Ce métamodèle est destiné à un domaine d'affaires particulier, celui de la conception des applications logicielles de gestion d'entreprises.

Le chapitre 6 introduit le concept du diagramme de composants d'affaires (BCM) ainsi que le profil UML2 permettant de le modéliser. De plus, une technique permettant, à l'aide de règles de transformation, de transformer le BPM en BCM est proposée.

Le chapitre 7 présente une nouvelle technique de construction du modèle des cas d'utilisation (UCM) basée sur le diagramme d'activité UML2. De plus, le profil UML2 de l'UCM ainsi que la technique permettant de transformer le BPM en UCM sont proposés.

Le chapitre 8 présente un gabarit de rédaction de cas d'utilisation sous format textuel et une technique de rédaction de cas d'utilisation sous un format visuel à l'aide du diagramme d'activité UML2. Un profil du diagramme d'activité UML est conçu à cette fin. De plus, des

règles de transformation permettant de passer d'un cas d'utilisation textuel en un cas d'utilisation visuel sont proposées.

Le chapitre 9 présente une technique de transformation du cas d'utilisation visuel en diagramme de classe d'analyse.

Le chapitre 10 présente une étude approfondie sur l'applicabilité de la méthodologie de transformation du CIM vers le PIM. Un cas d'étude est développé à cette fin.

Le chapitre 11 est la conclusion de cette thèse. Les limites, les contributions et les avenues futures de recherches y sont présentées.

CHAPITRE 1

L'INGÉNIERIE DIRIGÉE PAR LES MODÈLES : MISE EN CONTEXTE DU CIM ET DU PIM

1.1 Introduction

Dans le but de mettre en contexte le CIM et le PIM, ce chapitre vise à décrire, de manière globale, ce en quoi consiste l'ingénierie des modèles et, de manière spécifique, ce en quoi consiste l'architecture dirigée par les modèles ainsi que les technologies qui y sont reliées. Pour ce faire, il est nécessaire d'y présenter une revue de littérature et de s'interroger en premier lieu sur ce qu'est un modèle.

1.2 Qu'est-ce qu'un modèle ?

Il est difficile de trouver une définition universelle de ce qu'est un modèle. Le petit Larousse édition 2010 ne présente pas moins de huit définitions qui se résument dans les points suivants : un modèle est une abstraction de quelque chose qui a une existence réelle, il est différent de ce qui représente et peut être utilisé pour simplifier la compréhension de cette chose. Ce résumé introduit deux notions : la première rapporte que le modèle se distingue de ce qui est modélisé et la deuxième met l'emphase sur l'abstraction des détails. Ceci s'accorde avec la proposition de Stachowiak (Stachowiak, 1973) qui suggère trois caractéristiques principales que doit posséder un modèle :

1. Représentation : un modèle est toujours basé sur son original.
2. Réduction : un modèle ne représente pas toutes les propriétés de son original, mais se concentre d'en sélectionner les plus pertinentes selon le contexte et le but de son utilisation.
3. Subjectivité : un modèle ne peut jamais avoir une relation bijective avec son original. Il doit être utilisé à la place de son original pour satisfaire certaines fins. Son interprétation doit se faire par rapport à son usage et à son but.

Les deux premières caractéristiques abordent la notion de projection. Cela implique qu'un modèle original est projeté et que certaines informations sont perdues lors de cette projection. Ceci se résume par la notion d'abstraction qui cherche à retenir les informations utiles dépendamment de l'objectif du modèle et du contexte dans lequel il sera utilisé. La troisième caractéristique rejoint l'idée du pragmatisme dans l'utilisation des modèles qui se résume comme suit : un modèle est une information sur quelque chose qui a du contenu et du sens, créé par quelqu'un (créateur) et destiné à quelqu'un (utilisateur) pour satisfaire un but précis dans un contexte particulier.

Minskey (Patrick Coquillard et David R. C. Hill, 1997) définit ainsi un modèle : pour un observateur B, un objet A* est un modèle d'un objet A, si B peut utiliser A* pour répondre à des questions qu'il pose à propos de A. Cette définition rejoint celle de Stachowiak en terme de l'abstraction et de la simplification de la réalité : le modèle A* est une représentation simplifiée de l'objet A parce que de nombreuses caractéristiques de A ne sont pas considérées dans A*. Malgré cette fonction sélective, A* permet toutefois de répondre à des questions qui se posent sur A.

Dans le même ordre d'idée, Bézivin et Gerbé (Jean Bézivin et Olivier Gerbé, 2001) mentionnent qu'un modèle est une simplification d'un système et doit permettre de répondre à certaines questions à la place de l'objet modélisé : '*A model is a simplification of a system built with an intended goal in mind. The model should be able to answer questions in place of the actual system*'.

L'OMG (Joaquin Miller et Jishnu Mukerji, 2003) définit un modèle comme : '*a formal specification of the function, structure and or behaviour of a system'*. Le point central de cette définition est que les spécifications doivent être formelles, dans le sens que les éléments de modélisation doivent avoir une sémantique non ambiguë.

Dans le domaine du génie logiciel, un modèle peut être utilisé par différents individus dans différents espaces technologiques. Ainsi, un bon modèle devient un outil de communication

(Mellor Stephen J et al., 2004). Afin d'être interprété correctement, il est nécessaire que le langage de modélisation utilisé soit compris sans ambiguïté et ce, quelque soit le type de modèle et d'intervenants. En d'autres mots, ce langage de modélisation doit avoir une syntaxe et une sémantique permettant à un ordinateur de l'interpréter automatiquement.

L'objectif de la modélisation est de réduire la complexité du monde réel par abstraction des propriétés sélectionnées. Les modèles doivent être moins riches en termes de connaissances ou d'informations que l'objet modélisé. Ils sont donc des approximations des objets représentés. Un modèle n'est qu'une représentation simplifiée du réel se réalisant par une sélection de caractéristiques pertinentes à un contexte et une problématique donnés.

1.3 Taxonomie des modèles

Les définitions présentées sont larges et ne précisent pas la nature des modèles et des systèmes modélisés. Il est donc essentiel de faire une distinction entre les différents types de modèles notamment dans le contexte de transformation des modèles.

Dans ce sens, Mellor *et al.,* (Mellor Stephen J et al., 2004) distinguent trois types de modèles selon leur degré de précision et leur usage : esquisse, plan et exécutable.

- Un modèle de type esquisse n'est ni précis ni complet : il peut être réalisé rapidement pour des évaluations préliminaires.
- Un modèle de type plan doit être suffisamment précis afin de permettre la spécification d'un des modèles du système.
- Un modèle de type exécutable doit être précis et non ambiguë de manière à être traité et transformé en code source par un outil logiciel.

Cette distinction met en évidence que le degré de précision d'un modèle dépend de son usage.

Puccia et Levins (Charles J. Puccia et Richard Levins, 1985) ont identifié trois critères considérés comme des principes fondamentaux de la modélisation : généralité, précision et réalisme.

- La généralité réfère à la taille et la portée du système modélisé : un modèle d'une partie d'un objet est beaucoup plus spécifique que celui qui couvre tous les aspects de cet objet. Cependant, l'effort de la modélisation augmente énormément avec la généralité.
- La précision est une mesure de la granularité ou le degré de l'abstraction d'un modèle. Une précision élevée indique l'exactitude tandis qu'une précision faible indique la simplification. La précision peut être réduite par l'élimination des variables inutiles.
- Le réalisme est la mesure de l'authenticité du modèle par rapport à l'objet modélisé.

Puccia, Levins, Lee *et al.,* (Mark Lee, 2000) considèrent que la combinaison, deux-à-deux, de ces caractéristiques permet de découvrir trois types de modèles : *hard*, *soft* et *qualitative*.

1. réalisme et précision : ils caractérisent les modèles de type *hard*. Ceux-ci sont des modèles exacts et précis et leurs domaines d'application, comme les sciences physiques, sont souvent détaillés et sélectifs. En général, ces modèles sont bien acceptés dans les domaines qui ont bien spécifié leurs frontières. Ils s'appuient souvent sur des théories et leur degré de fiabilité ou de confiance est élevé.
2. généralité et précision : ils caractérisent les modèles de type *soft*. Les systèmes qui impliquent généralement des actions humaines comme les modèles écologiques, économiques ou sociaux sont des systèmes de types *soft*. Les modèles qui sont construits pour résoudre les problèmes de ces systèmes sont des modèles de type *soft*. Ceux-ci souffrent fréquemment d'un manque de données et leurs théories sous-jacentes sont souvent faibles. En général, quand le rôle de l'humain augmente dans un système de type *hard*, les mécanismes internes deviennent moins formulés et prennent les caractéristiques d'un système de type *soft*.
3. généralité et réalisme : ils caractérisent les modèles de type *qualitative*. En effet, si nous insistons sur des niveaux élevés d'authenticité (réalisme) dans des systèmes à grande échelle, la modélisation de ces systèmes mène au compromis de sacrifier la précision. Les

modèles de type *qualitative* sont une forme de modèles abstraits qui visent à capturer les aspects fondamentaux sans se soucier des détails.

Dans le génie logiciel, les modèles de types *hard* sont ceux qui sont utilisés dans le raisonnement sur la programmation, les systèmes électroniques et le matériel. La plupart des méthodologies de développement des processus, des exigences, de conception et d'analyse doivent être considérées comme des modèles de type *soft* en raison de leur incorporation de l'activité humaine. Quant aux modèles de type *qualitative*, ils offrent l'opportunité d'avoir un nouveau type de modèles.

Ludewing (Ludewig, 2003) classifie les modèles en deux types : *descriptif* et *prescriptif*. Quand un modèle peut refléter un objet existant, comme une photo, il s'agit d'un modèle *descriptif*, et dans le cas où un modèle est utilisé comme un cahier de charge ou un document des exigences logicielles, il s'agit d'un modèle *prescriptif*. En fait, le modèle *descriptif* cherche à décrire un système en utilisant des informations accessibles facilement et rapidement, tandis que le système *prescriptif* a pour mission de spécifier les caractéristiques de ce système modélisé. Quand un architecte de logiciels conçoit l'architecture d'un système existant, et puis ajoute certaines modifications qu'il propose à son client, le modèle est d'abord *descriptif* puis devient par la suite *prescriptif*. Ludewing qualifie ce passage du modèle *descriptif* à celui de *prescriptif* de modèle transitoire.

Dans le domaine de l'ingénierie du logiciel, l'OMG avec son approche MDA (Joaquin Miller et Jishnu Mukerji, 2003) classifie quatre types de modèles qu'il préconise pour la construction des logiciels : CIM, PIM, PSM et Code (que nous détaillerons plus loin dans ce chapitre). Un modèle d'un haut niveau d'abstraction contient moins de détails qu'un modèle de faible niveau d'abstraction. Dans MDA, il faut commencer par la création d'un modèle de haut niveau d'abstraction, qui sera transformé progressivement en des modèles de faible niveau d'abstraction. C'est de cette manière que les modèles dirigent la création des systèmes logiciels.

1.4 Modèles et métamodèles

Un métamodèle décrit de manière abstraite une structure possible de modèles. Il définit les concepts d'un langage de modélisation et de leurs relations, ainsi que les contraintes et les règles de modélisation. Plus précisément, un métamodèle définit la syntaxe abstraite et la sémantique statique d'un langage de modélisation.

Un modèle est une instance d'un métamodèle. Un langage de métamodélisation est nécessaire pour décrire un métamodèle qui à son tour est décrit par un méta-métamodèle. En d'autres mots, un métamodèle est un modèle du langage de modélisation (Mellor Stephen J et al., 2004; Seidewitz, 2003).

Markus Völter *et al.,*(Markus Völter et al., 2006) présentent la relation entre le modèle, le métamodèle et le monde réel. La Figure 1.1 illustre cette relation. Le métamodèle décrit les éléments du modèle qui à leur tour décrivent les éléments du son monde réel.

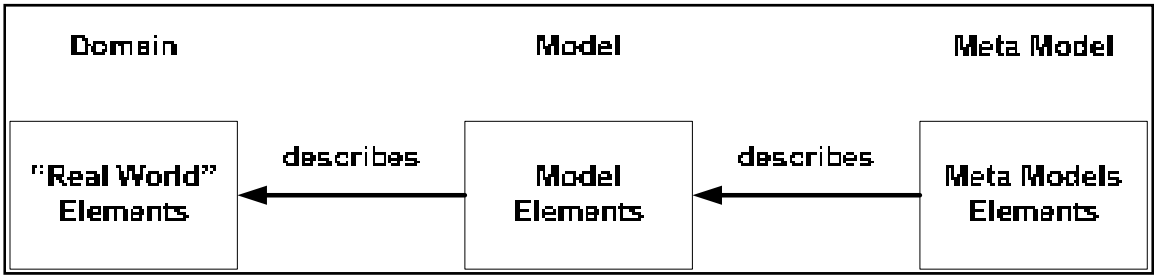


Figure 1.1 Relation entre modèle, métamodèle et monde réel
Empruntée de (Markus Völter et al., 2006)

Cette liaison entre le métamodèle et le langage de modélisation conduit à l'identification d'une nouvelle relation, la conformité d'un modèle vis-à-vis de son métamodèle. Cette relation permet d'assurer qu'un modèle est correctement construit et qu'il est possible de lui appliquer des transformations automatisées (Jean Bézivin et al., 2005).

Un modèle est conforme à un métamodèle et ce dernier peut aussi être un modèle conforme à un deuxième métamodèle. En théorie, cette abstraction en cascade peut continuer jusqu'à

l'infini, mais en pratique d'autres mesures ont été prises. L'OMG (OMG, 2010a) a défini une architecture à quatre niveaux de modélisation présentée dans la Figure 1.2. Il est à noter que cette architecture est dédiée uniquement au domaine du génie logiciel.

- **M0** : ce niveau décrit les instances (données) du logiciel lors de son exécution.
- **M1** : ce niveau contient les modèles qui représentent les données du niveau M0.
- **M2** : ce niveau définit les éléments de modélisation (la structure et la sémantique du langage utilisé) du niveau M1. Les éléments du niveau M1 sont des instances des éléments du niveau M2.
- **M3** : ce niveau permet de définir les modèles au niveau M2. La particularité du niveau M3 réside en sa capacité de se décrire et de s'instancier à partir de lui-même. Il n'existe pas de niveaux supérieurs de modélisation autre que le niveau M3.

Cette architecture à quatre niveaux présente plusieurs avantages (David S. Frankel, 2003; Mellor Stephen J et al., 2004; OMG, 2010a). Le fait qu'il existe un méta-métamodèle unique permet de définir tous les langages de modélisation dans un cadre unifié. La relation de conformité permet de faire face à la diversité des métamodèles, de les structurer dans un cadre qui permet de relier plusieurs métamodèles entre eux, d'ajouter des métamodèles, d'échanger des modèles et des métamodèles. Elle permet aussi de faciliter la transformation et l'intégration de modèles, appartenant à différents langages de modélisation.

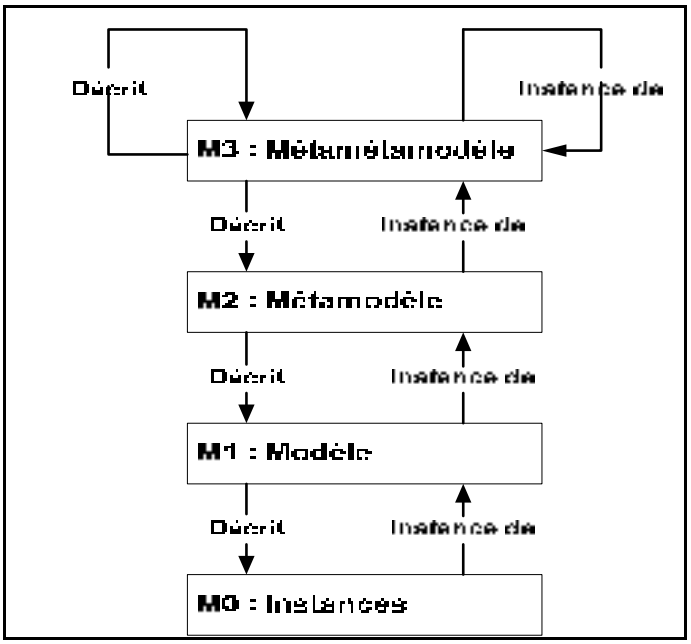


Figure 1.2 Les quatre niveaux de l'OMG
Empruntée de (Markus Völter et al., 2006)

L'architecture de métamodélisation la plus connue est celle de l'OMG qui repose sur un méta-métamodèle unique qui est le MOF (*Meta Object Facility*).

1.5 Ingénierie dirigée par les modèles (IDM)

L'ingénierie dirigée par les modèles (IDM) est une approche de développement logiciel qui focalise sur la création et l'exploitation des modèles plutôt que sur le code source (Kent, 2002). L'IDM consiste à définir un ensemble de modèles dans différents niveaux d'abstraction ainsi qu'un ensemble de règles de transformation permettant le passage d'un modèle source à un modèle cible. Les règles de transformation sont responsables de préserver la cohérence entre les différents modèles.

1.5.1 Le défi de l'IDM

Dans les années 1990, la communauté du logiciel a été influencée principalement par trois approches de développement logiciel : la méthode CASE (*Computer Aided Software Engineering*), les langages de quatrième génération (4GLs) et l'approche orientée-objet.

La méthode CASE et les outils logiciels qui la supportent étaient dispendieux et les approches propriétaires ont fait face à la présence croissante des approches ouvertes et gratuites. L'approche orientée-objet n'a pas tenue ses promesses (Jorgensen et al., 2002; Scholtz et al., 1993) mais a quand même réussi à remplacer les anciennes générations des langages de programmation. De ce fait, les outils de modélisation orientée-objet, profitant de la notation du standard UML et du désintérêt des deux approches CASE et 4 GLs, ont retenu l'attention (Thomas Stahl et al., 2006).

Les outils logiciels supportant le langage UML comme Rational Rose, dans ses versions 1997 et 2000, ne permettaient pas de synchroniser complètement le code source avec les modèles mais représentaient une méthode efficace pour générer de la documentation. C'est dans cette perspective que l'IDM vient combler cette faiblesse en augmentant le niveau d'abstraction afin de mettre l'accent sur les modèles. Le développement logiciel, d'après cette approche, se réalise en modélisant le logiciel à différents niveaux d'abstraction. À partir de règles de transformation spécifiques, ces modèles, dont les langages respectifs ont été rigoureusement définis à partir d'un méta-métamodèle unique, peuvent être transformés successivement du plus haut au plus bas niveau d'abstraction dans le but de générer le code source automatiquement. En somme, les modèles deviennent exécutables.

1.5.2 Les avantages de l'IDM

L'avantage visé par l'IDM est l'augmentation de la vitesse du développement logiciel grâce à l'automatisation : le code source peut être généré à partir des modèles en utilisant une ou plusieurs étapes de transformation. Ceci permet de réaliser un logiciel beaucoup plus

rapidement. La durée allouée au développement ainsi qu'à la validation du produit logiciel en est donc raccourcie (Chris Raistrick et al., 2004; Selic, 2003; THE MIDDLEWARE COMPANY, 2003). Le site web de l'OMG (Object Management Group, 2010) expose plusieurs cas de projets logiciels qui se sont basés sur une architecture dirigée par les modèles. L'entreprise ABB (Interactive Objects Software GmbH) est un cas réussi parmi d'autres qui, selon Andreas Blaszczyk, de chez ABB a résumé : "*The code quality produced by the ArcStyler's model-based generation is consistent and clean, reducing the number of programmers and test engineers required to develop and maintain the test environments by approx. 45% compared to a conventional development approach.*". Dans ce contexte, les travaux de Parastoo *et al.,* (Mohagheghi, Dehlen et Neple, 2009; Parastoo Mohagheghi et Vegard Dehlen, 2008) résument les avantages de l'IDM en focalisant sur la qualité des modèles.

- Traçabilité et synchronisation entre les modèles et le code source : dans une approche traditionnelle de développement logiciel, les modèles sont réalisés en marge de leur implémentation et le code source est généré manuellement, ce qui est demandant en temps. Ceci pose un problème majeur lors des modifications dans le code. En effet, elles sont rarement accompagnées d'une mise à jour dans les modèles (Chris Raistrick et al., 2004). L'IDM dans ce sens améliore la traçabilité et la synchronisation entre les modèles.
- Meilleure réutilisation des modèles : une fois définie, l'architecture du logiciel et les modèles peuvent être réutilisés dans d'autres systèmes (Mellor Stephen J et al., 2004).
- Logiciel de qualité : le code source généré à partir des modèles est, en principe, sans erreur parce que les modèles ont été testés et validés (ex. architecture JEE, .NET. etc). Par conséquent, la qualité du logiciel devient dépendante de l'implémentation des concepts du domaine d'affaires plutôt que des éléments de l'architecture logicielle (Selic, 2003).
- Séparation des préoccupations : la réutilisation des modèles offre l'avantage de focaliser davantage sur les concepts liés aux domaines d'affaires plutôt qu'aux concepts liés à leur implémentation (Chris Raistrick et al., 2004; Selic, 2003).

En résumé, l'IDM vise à accroître la productivité du développement logiciel.

En revanche, il est important de mentionner que l'IDM reste une approche complexe qui sera maitrisée que par des spécialistes en architecture de logiciel. De plus, la difficulté de maintenir la cohérence entre les différents modèles représente un défi majeur.

1.6 Architecture dirigée par les modèles

En 2001, l'OMG a lancé une nouvelle approche baptisée *Model Driven Architecture* (MDA) (Joaquin Miller et Jishnu Mukerji, 2003) qui est une variante particulière de l'IDM. L'essence de MDA est la création d'architectures logicielles exécutables dirigées par les modèles plutôt que l'écriture manuelle du code source. Le principe clé de MDA consiste à s'appuyer sur les standards UML pour décrire séparément les modèles des différents niveaux du cycle de développement logiciel. Les objectifs majeurs de MDA sont la portabilité, l'interopérabilité, la réutilisation, la maintenance et l'augmentation de la productivité.

Plus précisément, MDA préconise l'élaboration des quatre modèles suivants (Joaquin Miller et Jishnu Mukerji, 2003) :

- **CIM (Computation Independent Model)** est un modèle où aucune considération informatique n'y apparaît. Il modélise le domaine d'affaires et les exigences du système, en général par l'utilisation de modèles distincts (ex. processus d'affaires, cas d'utilisation, etc.). Il montre le système dans l'environnement organisationnel dans lequel il sera exécuté. Son but est d'aider à la compréhension du problème d'affaires, mais aussi fixer un vocabulaire commun pour les analystes et les architectes d'affaires. Les exigences exprimées dans le CIM doivent être traçables dans le PIM et le PSM. Il est à noter que le modèle CIM n'a été défini et ajouté à l'approche MDA qu'en 2003. L'OMG vise, par l'introduction du CIM, à combler le fossé entre les analystes d'affaires et les architectes du logiciel.
- **PIM (Platform Independent Model)** est un modèle qui décrit les concepts d'affaires du système sans montrer les détails de son utilisation sur une plateforme technologique

particulière. Le PIM doit être raffiné par les détails d'une ou plusieurs architectures particulières pour obtenir un PSM.

- **PSM (Platform Specific Model)** est le modèle produit par la transformation du PIM. Il est adapté pour spécifier l'implémentation du système dans une seule et unique plateforme technologique.
- **Code** est la phase finale du développement soit la transformation de chaque PSM en code source. Le code source est considéré comme un modèle dans l'approche MDA.

En somme, le MDA prône l'utilisation de ces quatre modèles dans l'ordre suivant :

- réaliser le modèle des exigences CIM qui spécifie et représente le système dans son environnement;
- réaliser le modèle d'analyse PIM à partir des éléments du CIM;
- enrichir le PIM par des détails techniques relatifs aux choix de la plateforme d'exécution, pour obtenir le PSM;
- raffiner le PSM jusqu'à l'obtention du code source.

1.7 Les standards de l'OMG

1.7.1 MOF (Meta Object Facility)

Le MOF (*Meta Object Facility*) est un standard de l'OMG spécifié dans MOF 2.0 *Core specification* (OMG, 2004). Il représente la couche supérieure dans la hiérarchie des métamodèles de l'OMG et est situé au niveau M3, comme illustré à la Figure 1.2. Son but est de spécifier d'autres métamodèles, qui, à leur tour spécifient des modèles comme UML. Le MOF permet d'échanger et de sérialiser les modèles entre eux via le standard XMI (*XML Metadata Interchange*). La spécification de MOF fournit un ensemble d'objets génériques et abstraits et leurs associations ainsi qu'un ensemble de règles pour créer et manipuler des métamodèles.

1.7.2 MOF et UML

UML est une instance et une application de MOF (OMG, 2010a). La notation du MOF est une syntaxe concrète d'UML. UML existait avant le MOF et n'avait pas une définition formelle. Le MOF est arrivé plus tard pour définir formellement UML d'où les révisions majeures d'UML (OMG, 2010a; 2010b) basées sur le MOF.

Dans le contexte de développement logiciel, il n'est pas coutume de créer un nouveau langage pour modéliser un domaine particulier mais plutôt d'essayer d'étendre un langage existant pour le rendre suffisamment apte à modéliser les concepts de ce domaine. Dans ce sens, UML, en s'appuyant sur le MOF, offre deux alternatives : extension basée sur le métamodèle et extension à l'aide des profils UML2.

1.7.3 Extension basée sur le métamodèle

Ce type d'extension étend le métamodèle d'UML. À cette fin, il faut se baser sur le MOF pour pouvoir y rester compatible. Pour définir un nouveau type de classe UML, il faut créer une nouvelle classe dans le niveau M2 (Figure 1.2) qui hérite de Metaclass UML::Class comme illustré à la Figure 1.3.

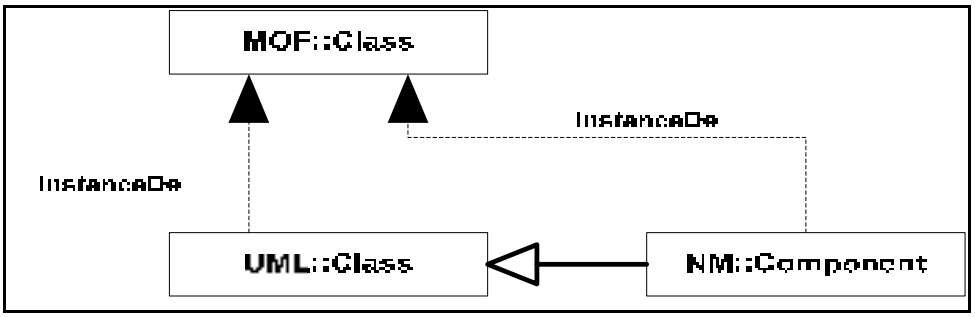


Figure 1.3 Adaptation d'UML à travers l'extension du métamodèle UML
Empruntée de (Markus Völter et al., 2006)

Dans la Figure 1.3 , un nouveau langage est défini, le NM::Component. C'est une sous classe de l'élément Class d'UML (UML::Class) et en même temps une instance de l'élément Class de MOF (MOF::Class). Cette nouvelle classe peut se différencier par l'ajout d'attributs, d'opérations et de contraintes. Ce type d'adaptation de métamodèle n'est pas restreint à UML, mais peut aussi s'appliquer à n'importe quel langage de modélisation basé sur le MOF.

1.7.4 Extension à l'aide des profils UML2

Un profil UML est un concept qui permet d'adapter les éléments du métamodèle UML à un domaine spécifique (technique, affaires, plateforme, etc.). Essentiellement, cela signifie l'introduction de nouveaux types d'éléments de modélisation en se basant les éléments originaux d'UML. À titre d'exemple, SysUML est un profil UML dédié au domaine de l'ingénierie des systèmes.

Ce mécanisme de profilage ne permet pas de modifier des métamodèles existants ou d'en créer des nouveaux. Il permet uniquement d'adapter ou de personnaliser un métamodèle existant. En fait, il n'est pas possible d'enlever des contraintes qui s'appliquent à un métamodèle mais il est possible d'en rajouter des nouvelles qui sont spécifiques au profil.
Un profil UML est défini par quatre types d'artefacts : *metaClass, stereotype, tagged values* et *constraint*.

- *metaClass* est une classe de profil qui peut être étendue par un ou plusieurs *stereotypes*.
- stereotype est une classe de profil qui utilise le mot clé *«stereotype».* Elle définit l'existence d'une *metaClass* étendue. Le *stereotype* a des propriétés appelées *tag definitions.* Quand un *stereotype* est appliqué à un élément du modèle UML, les valeurs de ses propriétés sont désignées comme des *tagged values.*
- *tagged values* sont les propriétés du *stereotype*. Ils ont un nom et un type.
- *constraint* décrit des restrictions sur les éléments du modèle stéréotypés. Il est souvent décrit en utilisant le langage OCL (*Object Contsraint Language*).

En somme, un profil UML est un mécanisme qui permet de spécialiser une variante d'UML à un domaine particulier. Il existe plusieurs profils standardisés par l'OMG comme le profil de CORBA, de SoaML (langage de modélisation de l'architecture orientée service), des EAI (*Enterprise Application Integration*) et plusieurs autres.

1.8 Transformation des modèles

La deuxième préoccupation clé de l'IDM consiste à rendre opérationnel les modèles à l'aide de transformations. Cette notion est au centre de l'approche MDA. En effet, MDA repose sur le principe de la création d'un modèle CIM pouvant être raffiné en PIM, ensuite au PSM pour enfin générer automatiquement le code source. En conséquence, il y a un besoin de techniques et d'outils permettant cette transformation de modèles.

Dans cette section, nous explicitons le mécanisme, les différentes techniques de transformation de modèles et les langages importants permettant l'exécution des opérations de transformation.

1.8.1 Définition

Tratt (Tratt, 2005) définit, d'une manière très large, la transformation des modèles comme "*a program that mutates one model into another*". L'OMG (Joaquin Miller et Jishnu Mukerji, 2003), dans le contexte de MDA, donne la définition suivante "*the process of converting a model into another model of the same system*". Kleppe *et al.,* (Anneke Kleppe, Jos Warmer et Wim Bast, 2001) proposent une définition plus complète "*A transformation is the automatic generation of a target model from a source model, according to a transformation definition. A transformation definition is a set of transformation rules that together describe how a model in the source language can be transformed into a model in the target language. A transformation rule is a description of how one or more constructs in the source language can be transformed into one or more constructs in the target language*". Cette dernière définition est très générale et couvre un large éventail d'activités où la transformation de

modèles peut être utilisée. En effet, afin de soutenir cette activité de transformation de modèles, le besoin de langages de transformation qui décrivent la manière de spécifier cette transformation, de méthodes et d'outils logiciels qui supportent cette transformation est nécessaire.

1.8.2 Taxonomie de transformation de modèles

Mens (Mens Tom et Van Gorp Pieter, 2006) a proposé une classification de transformation de modèles. Cette section présente les éléments essentiels de cette classification.

1.8.2.1 Transformation endogène versus transformation exogène

Pour effectuer une transformation de modèles, il est important que ces modèles soient exprimés par un certain langage de modélisation, par exemple le formalisme UML. Une distinction peut être faite en fonction du langage de modélisation utilisé pour exprimer les modèles sources et les modèles cibles impliqués dans une transformation : une transformation endogène concerne des transformations entre des modèles exprimés avec le même langage (s'appuyant sur le même métamodèle) et une transformation exogène concerne des modèles exprimés avec des langages différents (s'appuyant sur deux différents métamodèles).

- Un exemple typique de transformation endogène est la transformation d'un modèle de diagramme d'activités en un modèle de classe. Ces deux modèles sont exprimés en UML.
- Un exemple typique de transformation exogène est la transformation d'un diagramme d'activités exprimé avec le formalisme UML en un modèle BPMN (*Business Process Model Notation*).

1.8.2.2 Transformation horizontale versus transformation verticale

Le niveau d'abstraction est un critère important permettant de classifier les transformations de modèles. Une transformation entre deux modèles du même niveau d'abstraction est dite horizontale. Un exemple typique est une transformation entre deux PSM.

Quand la transformation concerne deux modèles de deux différents niveaux d'abstraction, il s'agit d'une transformation verticale. La transformation du CIM en PIM est un exemple typique de ce type de transformation verticale.

1.8.3 Techniques et approches de transformation

Cette section décrit les techniques et les approches de transformation de modèles proposées par différents auteurs et introduit d'une manière brève chacune de ces techniques.

1.8.3.1 Langage impératif versus langage déclaratif

Un des premiers critères à vérifier est de savoir si le langage de transformation repose sur un langage déclaratif ou un langage opérationnel.

Les langages impératifs ou opérationnels spécifient un flux de contrôle séquentiel. Ils fournissent des moyens capables de décrire comment le langage de transformation devrait s'exécuter. Les instructions et les concepts de ce langage sont similaires aux langages de programmation orienté objet comme le java ou le C++. Ils offrent un haut niveau de contrôle, ce qui donne de la flexibilité aux programmeurs et permet une implémentation efficace.

Les transformations à l'aide de ces langages sont décrites comme une séquence d'actions, ce qui est bien utile dans le cas où l'ordre des règles de transformation devrait être contrôlé explicitement (Geert Jan Bex, Sebastian Maneth et Frank Neven, 2002). En effet, l'approche impérative est plus appropriée grâce aux notions de séquence, de sélection et d'itération.

Les langages déclaratifs ou relationnels ne garantissent pas un contrôle explicite. Au lieu de décrire comment la transformation doit être exécutée, l'accent est mis sur ce qui doit être transformé par l'opération de transformation. Cette approche décrit la relation entre le modèle source et le modèle cible et peut être interprété d'une manière bidirectionnelle. Les langages déclaratifs sont compacts et les descriptions des transformations sont courtes et concises.

Il existe des langages dits hybrides qui offrent à la fois les deux approches, impératives et déclaratives, comme le langage QVT (*Query View Transformation*) (Object Management Group, 2008).

1.8.3.2 Langages de transformation graphiques (Graph transformation)

Ce type de langages est bâti sur des fondements algébriques, notamment la grammaire des graphes. Il est une sous-catégorie des langages déclaratifs. La transformation graphique a d'intéressantes propriétés théoriques et est souvent utilisée dans des approches mathématiques. Les modèles sont interprétés comme des graphes et la transformation manipule des sous graphes.

1.8.3.3 Approche basée sur des templates

Les approches basées sur des *templates* sont utilisées dans la génération du code source à partir des modèles. La transformation du PSM en code source représente un bon exemple de cette approche. Les modèles contiennent un métaprogramme qui peut accéder au code source. Généralement, les langages basés sur des *templates* incorporent le patron de conception *visitor* (Matthias Biehl, Chen DeJiu et Martin Törngren, 2010) permettant ainsi de traverser la structure interne du modèle.

1.8.3.4 Langage visuel versus langage textuel

Un autre critère distinguant les langages de transformation est celui de la spécificité de la syntaxe concrète du langage. Il y a des langages de transformation textuels, comme ATL (Freddy Allilaire et al., 2006) et Kermeta (Zoé Drey et al., 2010), exigeant de préciser les transformations de modèles en utilisant une description textuelle. Il y a aussi des langages de transformations visuelles qui spécifient les transformations de modèle d'une manière visuelle comme tout langage de transformation graphique. Il est à noter que certains langages offrent les deux alternatives comme le langage QVT (Object Management Group, 2008).

1.8.4 Langages de transformation de modèles

Chaque approche de transformation de modèle requiert un langage de transformation. Dans cette perspective, Czarnecki (Czarnecki et Helsen, 2006; Krzysztof Czarnecki et Simon Helsen, 2003) a proposé une série de critères qui doivent caractériser un langage de transformation :

- doit être exécutable;
- doit avoir une implémentation efficace;
- doit être expressif et non ambiguë;
- doit fournir une description précise, concise et claire des règles de transformations;
- doit différencier les règles de sélection des éléments du modèle source et les règles de transformation.
- doit proposer des constructions graphiques (une notation visuelle est plus concise et intuitive qu'une notation textuelle);
- doit proposer la possibilité de faire des transformations composites à l'aide des opérateurs de séquences, de conditions et de répétions;
- doit offrir la possibilité de faire des transformations bidirectionnelles et une mise à jour incrémentale des transformations (possibilité de reporter les changements effectués sur un modèle source dans le modèle cible).

Depuis l'avènement de l'approche MDA, le QVT et l'ATL sont les langages les plus utilisés et les plus complets à ce jour.

1.8.4.1 QVT (Queries, Views, Transformations)

Le QVT est un langage de transformation de modèles standardisé par l'OMG (Object Management Group, 2011). Il permet la transformation de modèle à modèle et du modèle vers du texte en utilisant particulièrement le métamodèle d'UML. Donc, il est possible de générer du code source ou même de la documentation grâce à ce langage.

QVT, comme le montre la Figure 1.4 , est constitué de quatre composantes. Trois de celles-ci sont des spécifications de langage dédié. Ces spécifications se nomment *Core*, *Relation* et *Operational Mapping*.

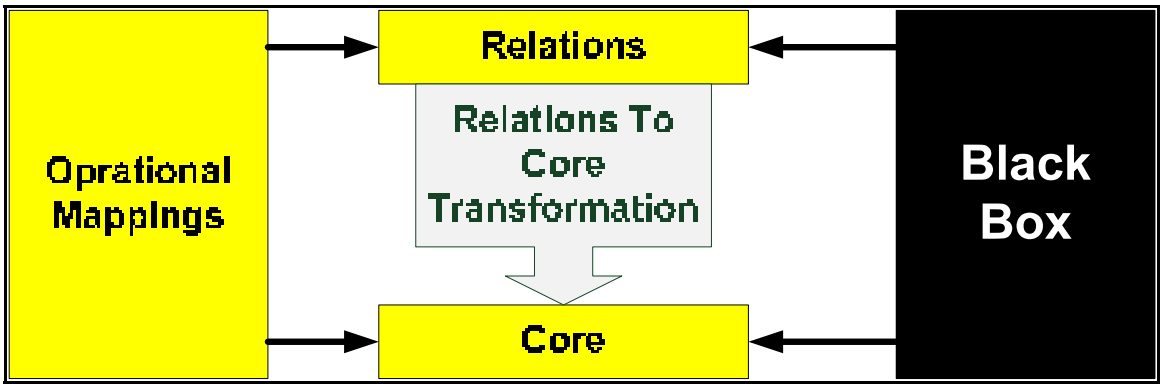


Figure 1.4 Relations entre les métamodèles de QVT
Empruntée de (Object Management Group, 2011)

- ***QVT relational*** est un langage déclaratif de haut niveau permettant de faire des transformations de modèle à modèle. Les transformations sont définies sous forme de règles qui après la transformation doivent être respectées totalement par le modèle cible. Une opération de transformation est une spécification d'un ensemble de relations entre le métamodèle source et le métamodèle cible, permettant ainsi de vérifier la correspondance, de faire respecter la cohérence et de synchroniser d'autres modèles candidats à cette même opération de transformation. Le QVT permet de faire des transformations bidirectionnelles et du *pattern matching* (filtrage par motif). Il supporte le filtrage complexe des éléments du modèle à l'aide de l'OCL. La traçabilité est créée implicitement et la sémantique est définie par le langage QVT *Core*.

- ***QVT* Core** est une version simplifiée du langage ***QVT Relational.*** Le *pattern matching* du langage *Core* est limité. Contrairement au langage *Relational*, les traces de la transformation ne sont pas effectuées automatiquement. Le programmeur doit expliciter ce qui doit être tracé. Un fait intéressant de l'interpréteur QVT est qu'il transforme les instructions de langage *Relational* en instructions de langage *Core* pour faciliter

l'interprétation. Ceci est analogue aux principes des machines virtuelles qui transforment les instructions d'un langage en instructions plus simples.

- ***QVT Operational*** est un langage de transformation impératif qui étend le langage QVT *Relational* avec des instructions impératives. Au lieu de déclarer comment le modèle source doit être, le programmeur déclare les transformations à appliquer. Le QVT *Operational* permet seulement des transformations unidirectionnelles et utilise en grande partie le langage OCL2. Il est possible d'utiliser presque toutes les fonctionnalités d'OCL. Par contre, l'utilisation d'OCL doit s'imbriquer dans des fonctions appelées *mappings*, *queries*, et *helpers* de QVT, un peu comme dans les langages traditionnels où les instructions doivent être contenues dans une méthode.

Le dernier composant du langage QVT est le *Black Box Implementation*. Il permet d'étendre le langage QVT avec des fonctionnalités qui seraient difficiles à construire à l'aide des langages proposés par QVT ou avec OCL.

1.8.4.2 ATL (Atlas Transformation Language)

ATL est un langage de transformation hybride inspiré du langage QVT. Il permet à la fois des constructions déclaratives et impératives. Toutefois, les auteurs d'ATL (Freddy Allilaire et al., 2006) recommandent l'utilisation du style déclaratif qui simplifie l'écriture des transformations.

Les transformations dans ATL sont unidirectionnelles mais il est possible de faire des transformations bidirectionnelles à l'aide d'un ensemble de règles implémentées à la fois dans les deux directions : le modèle source et le modèle cible.

Les transformations dans ATL se composent de trois modules : Header, Helpers et Rules.

- ***Header*** est utilisé pour déclarer des informations générales tels que le nom du module et de la transformation, des métamodèles source et cible, des bibliothèques utilisées dans la transformation.

- ***Helpers*** sont des instructions génériques basées sur l'OCL pour éviter la redondance du code de transformation.
- ***Rules*** sont des expressions décrivant la manière de générer les éléments du modèle cible (basé sur le métamodèle cible) à partir des éléments du modèle source (basé sur le métamodèle source).

Il existe une implémentation d'ATL dans l'IDE d'Eclipse dotée d'un moteur de règles responsable de la compilation et l'exécution des règles de transformation.

1.9 Mécanisme de transformation des modèles

Une transformation de modèles se réalise en général selon trois étapes (Olfa Djebbi, 2004) :

1. **Définition des règles de transformation :** cette étape consiste à mettre en correspondance les éléments du modèle source et les éléments du modèle cible. Il faut donc définir un métamodèle pour mettre en place des règles de transformation génériques. Les instances de ce métamodèle sont des spécifications (règles de transformation) de transformation entre deux métamodèles source et cible. Le processus de transformation prend en entrée un modèle source conforme au métamodèle source et produit en sortie, à l'aide des règles de transformation, un modèle cible conforme au métamodèle source.
2. **Expression des règles de transformation :** il faut un langage de transformation (exemple QVT ou ATL) capable de spécifier, filtrer et sélectionner les éléments de chacun des modèles source et cible pour ensuite les écrire dans un format interprétable par un moteur d'exécution de règles de transformation.
3. **Exécution des règles de transformation :** les règles de transformation, une fois spécifiées et exprimées, requièrent un moteur d'exécution. Celui-ci prend en entrée le modèle et son métamodèles source, les règles transformation ainsi que le métamodèle du modèle cible. Il produit en sortie le modèle cible.

La

Figure 1.5 résume les trois étapes du mécanisme de transformation de modèles à l'aide des règles de transformation.

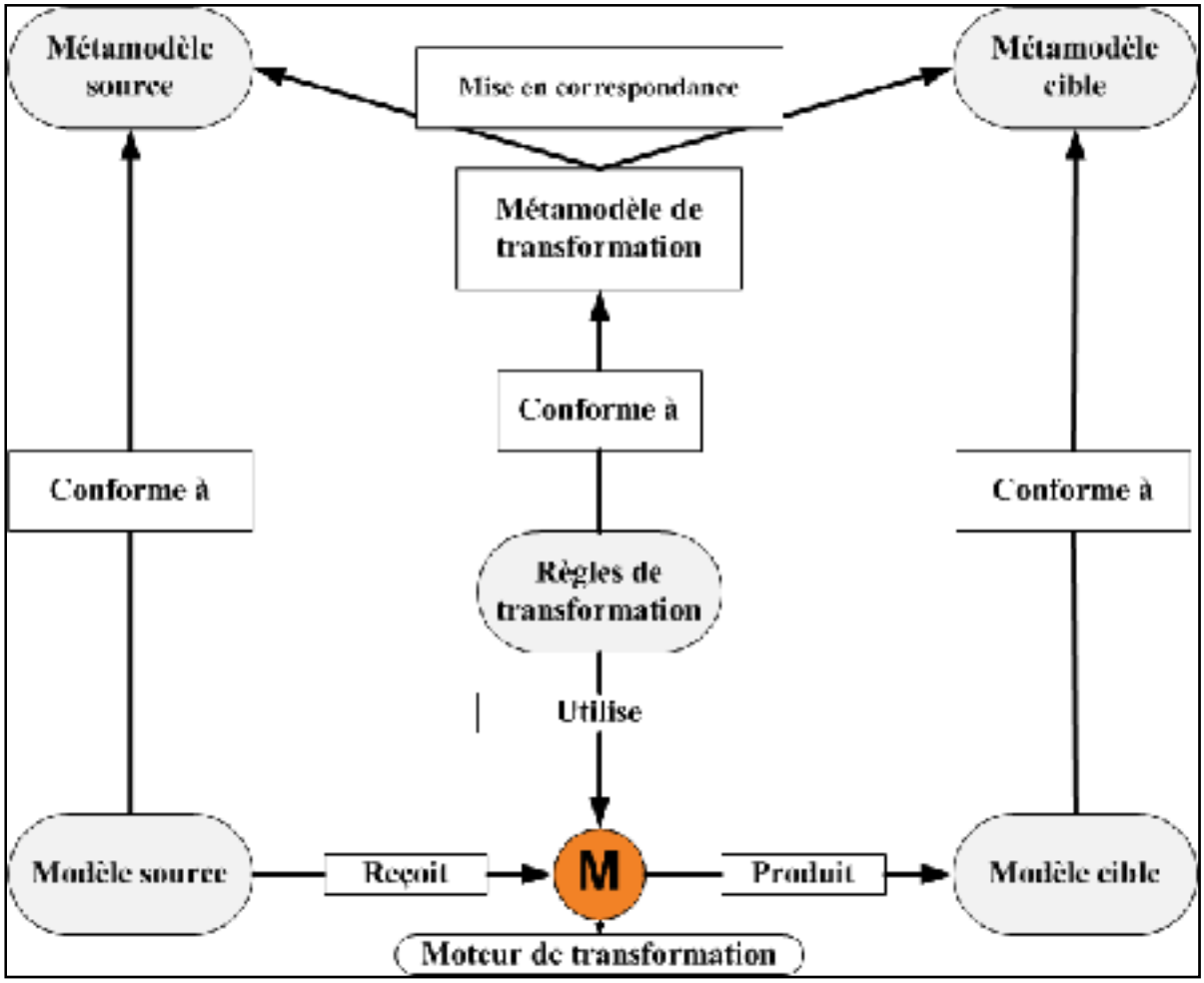


Figure 1.5 Mécanisme de transformation de modèles
Emprunté et modifié de (Olfa Djebbi, 2004)

1.10 Computation Independent Model (CIM)

Comme le définit l'OMG (Joaquin Miller et Jishnu Mukerji, 2003), le CIM est le modèle qui représente le plus haut niveau d'abstraction dans l'approche MDA. Son rôle principal réside dans la description du domaine d'affaires et la description des exigences à l'aide d'un langage compréhensible par les analystes d'affaires. En principe, le CIM doit être transformé

automatiquement en PIM. Sur ce dernier point, qui représente la problématique principale de cette thèse, nous exposerons les efforts effectués dans cette lignée de recherche.

Debnath *et al.,* (Narayan Debnath et al., 2008) considèrent que le formalisme UML n'est pas approprié à la construction du CIM. Cette inadéquation se justifie par la non familiarisation des analystes d'affaires, responsables de capturer les exigences avec le langage UML. Les auteurs proposent d'utiliser uniquement le langage naturel dans la phase de construction du CIM et ensuite, définir des règles de transformation pour générer le PIM sous forme de modèle de classe UML. Ils s'appuient donc sur deux modèles :

Language Extended Lexicon (LEL) : il s'agit d'une structure permettant de représenter les symboles importants de l'univers du discours. Le LEL est composé d'un ensemble de symboles caractérisés par des noms et un ensemble de synonymes, de plus que trois autres notions : *signs*, *notions*, et *behavioral responses* (Julio Cesar Sampaio do Prado Leite et Ann Paula M. Franco, 2003). Les symboles de LEL définissent des objets, des sujets, des phrases verbales et des états.

Scenario Model *:* il décrit les situations de l'univers des discours. Un scénario est relié au LEL et se compose d'un titre, d'un objectif, d'acteurs impliqués dans le scénario, de ressources et d'un ensemble d'épisodes (chaque épisode représente une action d'un acteur utilisant des ressources).

Ensuite, les auteurs dans (Narayan Debnath et al., 2008) ont défini une série de règles de transformation permettant de transformer les exigences écrites en format LEL en un diagramme de classe préliminaire. Cette transformation nécessite une intervention humaine pour compléter certains concepts et apporter quelques modifications.

Cette manière de construire le CIM en vue de le transformer en PIM concorde avec plusieurs travaux, en l'occurrence, ceux qui s'intéressent à transformer les exigences logicielles exprimées en langage naturel contrôlé au diagramme de classe UML (Ibrahim et Ahmad; Il-

yeol Song and Kurt Yano et Juan Trujillo, 2004; Sharma et al., 2009; Xiaohua Zhou et Nan Zhou, 2008).

Dans le cadre du projet MINT (*Modellgetriebene Integration von Informationssystemen*) qui consiste à fournir des méthodes pour intégrer les systèmes d'information avec les bases de MDA, les auteurs dans (Niels Streekmann et al., 2006) proposent de construire le CIM en utilisant deux modèles : un modèle de processus d'affaires modélisé avec le diagramme d'activité UML2 et un modèle de domaine qui incorpore toutes les informations nécessaires sur un domaine d'affaires particulier.

En vue de concevoir le CIM et le transformer en PIM, le modèle de processus d'affaires et le modèle du domaine s'appuieront sur deux types de patrons soit des patrons spécifiques au domaine (en cours de développement par les auteurs) et des patrons cognitifs (des gabarits que les humains utilisent pour la résolution/raisonnement des activités d'un problème (Karen M. Gardner et al., 1998)). Il est à noter, à la date de rédaction de cette thèse, que cette recherche ne mentionne pas les détails permettant de construire le CIM ainsi que la manière de le transformer en PIM.

Osis *et al.,* (Janis Osis, Erika Asnina et Grav, 2008) ont proposé une autre approche de transformation du CIM en PIM nommée *Topological Functioning Modeling for MDA* (TFMMDA). Dans le but de construire le CIM, cette approche repose sur deux activités essentielles, la première concerne l'analyse du problème (au niveau des affaires) et la seconde concerne l'analyse de la solution possible (niveau applicatif). Une fois les connaissances sur un système complexe acquises et connues, une fonction topologique appelée TFM pourra être composée. Elle modélisera les concepts et les liens entre les différents éléments constituant ce domaine d'affaires. Le TFM sera utilisé pour vérifier la cohérence des exigences fonctionnelles par rapport au domaine d'affaires. Ensuite, les caractéristiques du TFM seront associées aux objectifs d'affaires du système et les exigences fonctionnelles seront complétées par la découverte des acteurs pour finalement construire le modèle des cas d'utilisation. Les exigences sont écrites en langage naturel et la

transformation du CIM en PIM sera assurée par la transformation du langage naturel (exprimant le CIM) en diagramme de classe (exprimant le PIM) en utilisant la théorie des graphes. Les auteurs n'excluent pas l'intervention humaine durant le processus de conception du CIM et pendant sa transformation en PIM.

Pastor *et al.,* (Oscar Pastor et al., 1997) ont préconisé bien avant l'avènement de MDA une méthode de développement logicielle nommée *Object-Oriented Method* (*OO-Method*). Cette méthode est une compilation de modèles modélisant le système dans différents niveaux d'abstraction tout en distinguant entre l'espace du problème (plus haut niveau d'abstraction) et l'espace de solution (le plus bas niveau d'abstraction). Dans des travaux récents de Pastor *et al.,* (Oscar Pastor et al., 2008), l'emphase a été mis sur le rapprochement entre la méthode *OO-Method* et l'approche MDA comme le montre la Figure 1.6.

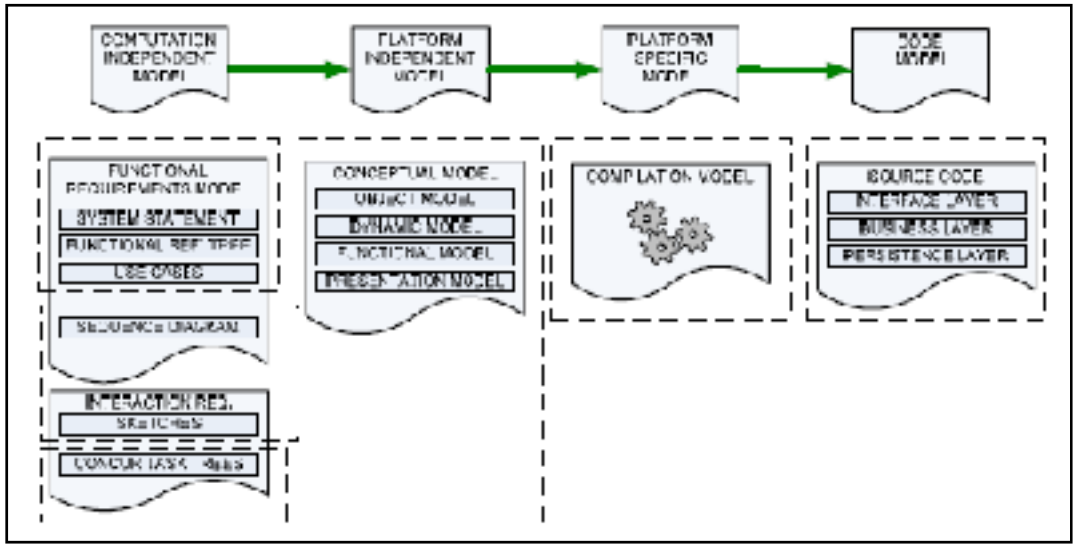


Figure 1.6 Compraison entre OO-Method et MDA
Emprunté de (Oscar Pastor et al., 2008)

Le modèle correspondant au CIM dans l'*OO-Method* est le *Functional Requirements* Model. Il est représenté par (E. Insfran, O. Pastor et R. Wieringa, 2002) :

Mission Statement qui décrit l'objectif du système, ses responsabilités importantes et les éléments dont il ne devra prendre en compte;

Function Refinement Tree qui traite les interactions externes du système selon les différents objectifs et domaines d'affaires;
Use–Case Model qui inclut les spécifications des cas d'utilisation et le modèle des cas d'utilisation.

En conclusion, les approches proposées pour la conception du CIM partagent la même idée : en dépit de la technique utilisée, l'activité de conception du CIM s'intéresse à capturer les exigences du système et à décrire les concepts d'affaires du domaine de problème.

1.11 Platform Independent Model (PIM)

Le PIM est par définition (Joaquin Miller et Jishnu Mukerji, 2003) un modèle indépendant de toute plateforme. Cette notion d'indépendance de plateforme est liée à la capacité du modèle à abstraire les caractéristiques d'une plateforme technologique particulière. Une définition assez générale de ce qu'est une plateforme est mentionnée dans (Joaquin Miller et Jishnu Mukerji, 2003): '*'a platform is a set of subsystems and technologies that provide a coherent set of functionality through interfaces and specified usage patterns*".

MDA recommande l'utilisation d'UML dans la conception des modèles CIM, PIM et PSM. Le PIM, selon MDA, décrit la structure d'information du système. Dans ce cas-ci, le diagramme de classe UML est le plus adapté à construire le PIM (Aditya Agrawal et al., 2002; Krzysztof Czarnecki et Simon Helsen, 2006; Oksana Nikiforova et Natalya Pavlova, 2008).

Il existe dans la littérature quelques techniques permettant de guider la construction des diagrammes de classe, en l'occurrence, le GRASP (*General Responsibility Assignment Software Patterns*) (Larman, 2004). En effet, une des pistes intéressantes que nous avons explorées et qui permet de faciliter la création du diagramme de classe est l'utilisation du concept d'archétypes et des patrons archétype.

Sur ce dernier point, Jim Arlow *et al.,* (Jim Arlow et Ila Neustadt, 2004) recommandent la génération du PIM en adaptant les patrons archétype tout en proposant quelques uns comme *Party, Product, Inventory, Order,* etc. Cette manière de faire est relativement facile et rapide en comparaison avec les méthodes traditionnelles de conception du diagramme de classe. Le processus d'adaptation des patrons archétypes à un domaine d'affaires particulier consiste, comme le montre la Figure 1.7 soit à ajouter de nouvelles entités au patron archétype soit d'en supprimer des entités, considérées comme optionnelles. De plus, si nous disposions du CIM, il pourrait guider le choix des patrons archétype les plus appropriés à la conception du PIM.

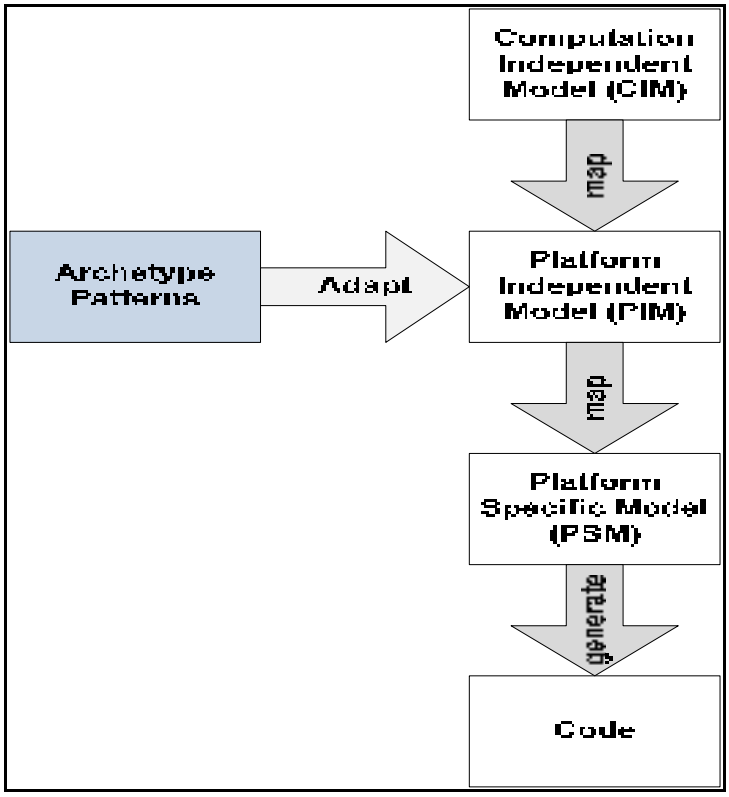


Figure 1.7 Adaptation du PIM au domaine d'affaires à l'aide des patrons archétype

Avant de préconiser l'utilisation du concept d'archétype dans la conception du PIM, il est important de donner une définition du concept d'archétype.

Le mot archétype vient du Grec *archetypo* et signifie « *original pattern* ». Dans (Jim Arlow et Ila Neustadt, 2004), un archétype est défini comme « *a primordial thing or circumstance that recurs consistently and is thought to be universal concept or situation* ». En effet, c'est au psychologue Carl Gustav que revient le concept de l'archétype (Carl Gustav, 1981). Il le définit comme une structure de représentation qui provient d'expériences humaines communes (*the collective unconscious*) délimitant un thème universel, mais figurée sous diverses formes symboliques.

Un des aspects intrigants dans la définition de Gustav est celui de la variabilité : les archétypes changent leurs formes afin de s'adapter à un contexte culturel spécifique tout en préservant leur sémantique. À titre d'exemple, l'archétype du méchant diffère d'une bande dessinée à l'autre et pourtant, le concept du méchant reste toujours le même, quelqu'un qui fait du mal et qui cause des ennuis à autrui.

Comme les archétypes sont des mécanismes structurant l'ensemble des processus psychiques de l'être humain et que celui-ci a été impliqué depuis des milliers d'années dans le domaine des affaires, il est tout à fait raisonnable d'observer l'apparition d'archétypes provenant de ce domaine. Par conséquent, il est possible d'en trouver des applications dans le domaine du développement de logiciels (Jim Arlow et Ila Neustadt, 2004). Par exemple, prenons l'activité de vente, une des plus vielles inventions humaines, qui en général, implique toujours le concept de produit, de prix, de vendeur et d'acheteur. Ceux-ci sont considérés comme des concepts fondamentaux (archétypes) et les relations entre eux mettent en évidence le concept du patron archétype. Par exemple, le prix et toujours associé à un produit.

Les systèmes logiciels reflètent le domaine d'affaires dans lequel ils opèrent. Il est ainsi possible d'en identifier des archétypes. Ces derniers sont appelés des archétypes d'affaires (*business archetypes*) et se définissent comme ''*A business archetype is a primordial thing that occurs consistently and universally in business domain and business software systems*''. Un bon exemple d'archétype d'affaires est le *Party* qui peut prendre la forme d'une personne

ou d'une organisation et que nous pouvons identifier pratiquement dans tous les domaines d'affaires.

Le concept de patron d'archétype d'affaires (*business archetype pattern*) (Jim Arlow et Ila Neustadt, 2004) réfère au collaboration entre les archétypes comme dans le cas de la collaboration de *Party*, *Product* et *Order*.

1.11.1 Caractéristiques d'un archétype

Dans la modélisation objet, il existe une différence fondamentale entre les classes d'analyse et les classes de conception. Les premières représentent une abstraction du domaine de problème et les secondes ont des spécifications complètes à un degré tel qu'elles peuvent être implémentées. De ce fait, il est important de mentionner que les archétypes sont toujours situés à un niveau supérieur de celui des classes d'analyse. D'un point de vue conceptuel, les archétypes sont consciemment reconnus pour capturer des concepts universels, tandis que les classes ne s'intéressent pas nécessairement aux concepts universels. D'autre part, d'un point de vue technique, un archétype génère une ou plusieurs classes d'analyse (Jim Arlow et Ila Neustadt, 2004).

Par analogie, cette différence entre les archétypes et les classes d'analyse s'applique aussi bien à la différence entre les patrons archétype et les patrons d'analyse.

Arlow *et al.,* (Jim Arlow et Ila Neustadt, 2004) ont présenté quatre caractéristiques spécifiant le concept d'archétype dans le contexte du domaine logiciel :

- ***Universal*** doit se produire continuellement dans le domaine d'affaires***;***
- ***Pervasive*** se produit à la fois dans le domaine d'affaires et dans le domaine logiciel. Celui-ci, est le principe de la convergence décrit dans (David A. Taylor, 1995) et repris dans (Richard Hubert, 2001);
- ***Deep history*** existe depuis que le principe d'affaires a commencé;

- ***Self-evident to domain expert*** qui n'est pas toujours le cas, mais demande parfois, de s'assurer que c'est vraiment un archétype.

Le terme archétype a été utilisé par Peter Coad (Peter Coad, Eric Lefebvre et Jeff De Luca, 1999) qui le définit comme ''a form from which all things of the same kind more or less follow''. L'utilisation des archétypes de Coad reste similaire aux archétypes de Arlow.

Coad *et al.,* (Peter Coad, Eric Lefebvre et Jeff De Luca, 1999), en se basant sur leur expériences de développement de logiciels, ont défini quatre archétypes : *Moment-Interval*, *Party-Place-Thing*, *Role* et *Description*. Chaque archétype est doté de responsabilités typiques. Les archétypes collaborent ensemble afin de proposer un modèle de classe d'analyse. Nous détaillerons dans le chapitre 4 les archétypes de Coad.

1.12 Problématiques non abordées dans la littérature

L'ingénierie dirigée par les modèles (IDM) dans sa variante MDA préconise l'utilisation des modèles et offre une première réponse aux quand, quoi et pourquoi modéliser. IDM vise à mettre en valeur les avantages intrinsèques des modèles, telles que productivité et prise en compte des plates-formes d'exécution. Le principe fondamental de MDA consiste en l'utilisation de modèles aux différentes phases du cycle de développement logiciel. Plus précisément, MDA prône l'élaboration des modèles CIM, PIM et PSM. Idéalement, selon MDA (Joaquin Miller et Jishnu Mukerji, 2003), le passage d'un modèle à l'autre doit se faire automatiquement à l'aide de techniques de transformation de modèles, le CIM en PIM, le PIM en PSM et le PSM en code source.

Toutefois, la littérature sur MDA n'aborde pas la question du comment réaliser les modèles CIM, PIM, PSM ainsi que comment passer d'un modèle à l'autre. Quoique plusieurs travaux de recherches se sont intéressés aux problématiques de conception du PIM, PSM et de la génération du code source à partir du PSM ainsi que les techniques permettant la transformation du PIM en PSM, peu d'auteurs ont abordé la problématique de conception du

CIM et sa transformation en PIM. Ce constat permet donc de citer, sur la base de la littérature, les problématiques reliées et ce qui reste à accomplir dans la transformation du CIM en PIM :

- L'absence de l'architecture du CIM. En d'autres mots, les éléments constituant sa structure et délimitant ses frontières avec le PIM.
- L'absence de l'architecture du PIM. En d'autres mots, les éléments constituant sa structure et délimitant ses frontières avec le PSM.
- La majorité des travaux de recherche (Janis Osis, Erika Asnina et Grav, 2008; Oscar Pastor et al., 2008) n'ont abordé le CIM que partiellement. Ils réduisent le CIM en un modèle des exigences exprimé par les cas d'utilisation et laissant de côté les autres artefacts comme le modèle des processus d'affaires et le modèle d'affaires permettant d'aligner les objectifs d'affaires avec le système d'information. Toutefois, le CIM (Joaquin Miller et Jishnu Mukerji, 2003) ne se réduit pas en un modèle de cas d'utilisation mais bien au-delà.
- L'absence d'une méthodologie permettant la conception du CIM.
- L'absence d'une technique permettant la transformation du CIM en PIM.

CHAPITRE 2

OBJECTIFS ET MÉTHODOLOGIE DE RECHERCHE

2.1 Objectifs de recherche

Cette section présente l'objectif principal et les sous objectifs de ce travail de recherche ainsi que la méthodologie de recherche.

L'approche MDA préconise l'utilisation des modèles et des métamodèles. Il est à rappelé que cette approche de développement logiciel consiste en la définition d'un ensemble de modèles successifs (CIM, PIM, PSM, Code) et de règles de transformation permettant de générer des modèles cibles à partir de modèles sources.

Comme ces différents types de modèles représentent différents niveaux d'abstraction d'un même système, un mécanisme de transformation devient nécessaire afin d'établir les liens de traçabilités ainsi que la manière de passer d'un type de modèle à l'autre. Les transformations de modèles préconisées par MDA sont essentiellement les transformations CIM vers PIM et PIM vers PSM. Quant à la génération du code à partir du PSM, elle n'est pas considérée comme une transformation de modèle à part entière.

Depuis l'avènement de MDA, plusieurs travaux ont abordé la problématique de transformation du PIM vers le PSM et du PSM vers le code mais très peu en revanche traitent de la transformation du CIM vers le PIM. Bien que la littérature présente quelques travaux reliés à cette question, il semble que peu de chercheurs se soient penchés sur les problèmes reliés à la transformation du CIM vers le PIM. Par conséquent, l'objectif principal de cette thèse est le suivant :

Objectif principal Concevoir une méthodologie qui permette la transformation du CIM vers le PIM.

Le CIM est représenté par le modèle des processus d'affaires, le modèle des cas d'utilisation et les détails de chaque cas d'utilisation, et le PIM est représenté par le modèle de classe d'analyse.

Cet objectif ne peut se concrétiser sans la réalisation d'un ensemble d'artefacts. En effet, la réalisation de ce processus de transformation méthodologique exige la définition précise de l'architecture et de la structure (l'ensemble des éléments constituant la structure des modèles CIM et PIM, les relations entre eux et leurs propriétés) des deux modèles CIM et PIM. L'approche MDA ne spécifie aucune méthodologie quant à l'élaboration de ces deux types de modèles ainsi que la manière de passer de l'un vers l'autre. De ce fait, plusieurs sous-objectifs s'imposent afin d'atteindre l'objectif principal de cette recherche.

1. Définition de l'architecture du CIM.
2. Définition de l'architecture du PIM.
3. Définition de la technique de construction du modèle des processus d'affaires.
4. Définition de l'architecture du modèle de composants ainsi que les règles de transformation permettant de transformer le modèle de processus d'affaires en modèles de composants d'affaires.
5. Définition de la technique de transformation du modèle de processus d'affaires en modèles de cas d'utilisation.
6. Définition du gabarit permettant de rédiger les détails de chaque cas d'utilisation.
7. Définition du modèle permettant de modéliser un cas d'utilisation visuel ainsi que les règles de transformation permettant de transformer un cas d'utilisation textuel en un cas d'utilisation visuel.
8. Définition des règles de transformation permettant la transformation d'un cas d'utilisation en diagramme de classe d'analyse.
9. Définition d'une technique permettant de fusionner l'ensemble des portions de diagramme de classe, générée à partir de chaque cas d'utilisation, en un seul diagramme de classe d'analyse.

10. Expérimentation de la méthodologie de transformation du CIM en PIM par une étude de cas.

Pour réaliser cette série de sous-objectifs, il est judicieux de définir une méthodologie qui facilite la réalisation de ce processus de transformation.

2.2 Méthodologie de recherche

Cette section présente un aperçu des activités de recherche à réaliser pour atteindre l'objectif et les 10 sous-objectifs de la proposition de recherche.

L'état de l'art a permis de constater l'absence d'une définition claire et précise de CIM ainsi que l'absence d'un cadre conceptuel incluant des techniques et des méthodes permettant sa transformation en PIM. Nous nous retrouvons ainsi à mener cette recherche avec un minimum de connaissances. Pour combler ce vide en reprenant les termes de Van der Maren (Jean-Marie Van Der Maren, 1996), la recherche exploratoire nous paraît la plus adéquate à clarifier cette problématique vue qu'elle sert à produire des connaissances sur des phénomènes inconnus. Pour ce faire, nous avons adopté deux approches menées en parallèle : d'une part une lecture large et approfondie de la littérature et, d'autre part, des entretiens non directifs.

- Une lecture large et approfondie de la littérature vise à informer sur les recherches reliées à ce domaine. Dans notre cas ce sont des recherches partielles et isolées, parce qu'elles n'abordent pas l'aspect global de la problématique de transformation du CIM vers le PIM mais plutôt la proposition d'éléments contribuant, à titre d'exemples, à la conception du CIM (ex. : processus d'affaires) et à la conception du PIM (ex. : archétypes).

- Des entretiens non directifs de type exploratoires visent à discuter davantage les aspects des questions de recherche dont certains sont absents de notre propre expérience et de nos lectures. L'objectif de cette approche ne consiste pas à valider nos idées préconçues, mais

bien à en construire de nouvelles et développer une réflexion issue des apports d'une série de rencontres et de discussions avec différents interlocuteurs tout au long du processus de la réalisation de cette thèse. Deux types d'interlocuteurs nous ont intéressé :

- Des spécialistes scientifiques : discussion avec des chercheurs et des professeurs universitaires qui sont directement liés, par leur travaux de recherche, à la question de cette thèse. Cette activité a été réalisée dans le cadre de conférences scientifiques, de correspondances directes et de participations à des forums de discussion.
- Des industriels expérimentés : discussion avec des industriels occupant des fonctions de directeurs de technologies de l'information, des analystes d'affaires, des architectes logiciels et des programmeurs qui s'intéressent à la modélisation UML. Cette activité a été menée, pendant mes mandats de consultation, dans différentes entreprises montréalaises (ex. : Almonix, Banque Nationale du Canada, Léger Marketing, Astra Zeneka, IBM, SAP).

En résumé, cette méthode exploratoire a permis de définir six étapes constituant la méthodologie de recherche :

Phase 1 : Exploration

Cette phase consiste à clarifier les objectifs de la recherche à travers la lecture de la littérature et les discussions menées avec différents interlocuteurs. Il s'agit de confronter la problématique de départ aux informations recueillies au cours de cette phase et de l'adapter éventuellement au développement de la réflexion issue des apports de celle-ci.

Phase 2 : Définition de l'architecture du CIM

Cette phase a pour objectif de définir les éléments constituant la structure du CIM. Celui-ci se compose de quatre modèles inter-reliés.

- Business Motivation Model (BMM);
- Business Process Model (BPM);
- Use Case Model (UCM);

- Business Component Model (BCM).

Phase 3 : Définition de l'architecture du PIM

De la même manière que la définition du CIM, l'objectif de cette phase est de définir les éléments constituant le PIM. Celui-ci se compose des quatre archétypes identifiés par (Peter Coad, Eric Lefebvre et Jeff De Luca, 1999) à savoir : *Moment-Interval*, *Party-Place-Thing*, *Role* et *Description*. L'idée de cette phase est de rapprocher la définition du PIM à celle du DNC (Domain Neutral Component). Celui-ci structure les quatre archétypes et vise à faciliter la création de n'importe quel modèle de classe d'analyse (PIM) dans le domaine des applications de gestion des entreprises (Eric Lefebvre, 2005).

Phase 4 : Détailler l'architecture du CIM

En se référant aux activités réalisées à la phase 2, il est important de noter que le BMM ne fera pas l'objet d'une étude détaillée dans cette thèse.

Cette phase 4 consiste en la réalisation des activités suivantes :

- Concevoir le modèle des processus d'affaires (BPM) :
  - Démontrer que le processus d'affaires élémentaire (EBP) est l'élément essentiel de la conception du BPM;
  - Définir des règles de cohérence de la conception du BPM;
  - Définir une procédure de création du BPM;
  - Proposer un profil UML est un métamodèle du BPM permettant de spécialiser les éléments nécessaires à sa création;
- Concevoir le modèle des composants d'affaires (BCM) :
  - Proposer le BPC (Business Process Component) et le BEC (Business Entity Component) en tant que deux éléments principaux dans la conception du BCM.
  - Définir les règles de cohérence de la conception du BCM.
  - Définir des règles de transformation permettant la transformation du BPM vers le BCM.
- Concevoir le modèle des cas d'utilisation (UCM) :
  - Définir la technique permettant de concevoir l'UCM à base du BPM.

- Proposer un gabarit, des règles de restriction, des règles de contrôle, un langage contrôlé et des patrons sémantiques pour écrire les cas d'utilisation.
- Proposer un profil UML, des règles de contrôle et des règles de restriction pour modéliser les cas d'utilisation sous format visuel.
- Proposer un mécanisme permettant, à l'aide des règles de transformation, de générer un cas d'utilisation visuel à partir d'un cas d'utilisation textuel.

Phase 5 : Définir le processus de transformation du CIM vers le PIM

Cette étape consiste, à l'aide des règles de transformation et des archétypes, à générer le modèle de classe d'analyse (PIM) à partir des cas d'utilisation visuels.

Phase 6 : Étude de cas

Cette phase consiste à réaliser une étude de cas permettant d'expérimenter l'ensemble des étapes nécessaires à la transformation du CIM en PIM.

La Figure 2.1 illustre la méthodologie de recherche avec à gauche les intrants de chaque phase et, à droite les extrants de chaque phase.

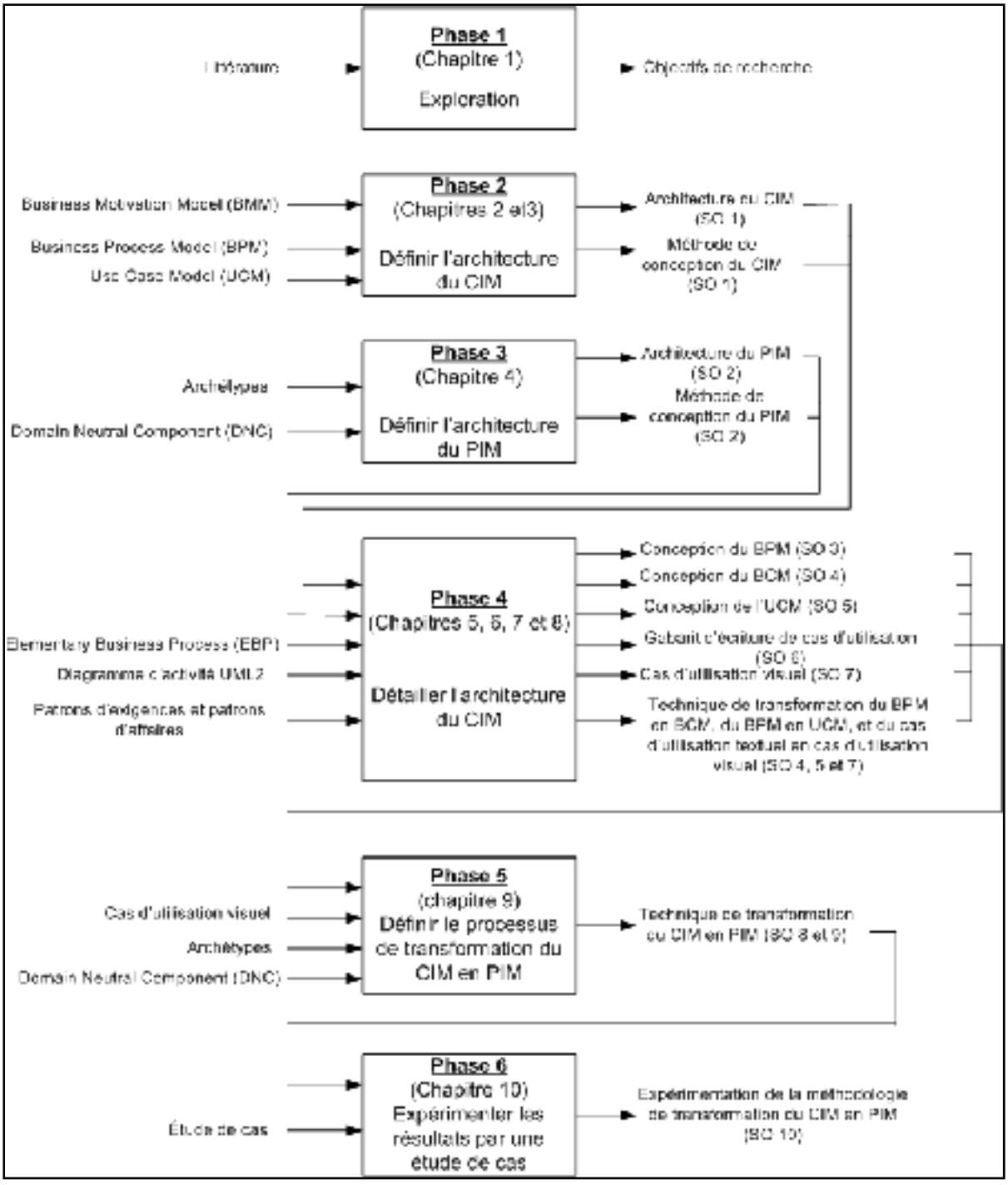


Figure 2.1 Les six phases de la méthodologie de recherche

CHAPITRE 3

CONCEPTION DE L'ARCHITECTURE DU <<COMPUTATION INDEPENDENT MODEL>> (CIM)

3.1 Introduction

Ce chapitre présente la phase 2 de ce projet de recherche, soit la définition de l'architecture du CIM.

Dans la définition de MDA (Joaquin Miller et Jishnu Mukerji, 2003), l'OMG s'est limité au « quoi? » plutôt qu'au « comment ? », laissant le soin de cette tâche aux éditeurs logiciels et aux chercheurs en génie logiciel. Bien que de nombreux efforts aient été investis dans la transformation du PIM à des niveaux inférieurs, il reste encore beaucoup de choses à faire au niveau du modèle CIM. Toutefois, avant de tenter de formaliser un ensemble de règles permettant la transformation du CIM en PIM, il est primordial de définir précisément ce qu'est un CIM. En d'autres mots, il est impératif de répondre aux questions suivantes : En quoi consiste l'architecture du modèle CIM ? Quels sont les éléments constituant sa structure ? Et comment procéder à son élaboration ?

Actuellement, il n'existe pas de définition formelle du CIM décrivant les éléments de sa structure auxquels les règles de transformation peuvent être appliquées et reliant ces éléments à l'intention d'affaires. Toutefois, une série de tentatives d'interprétation du CIM et du PIM sont résumées dans (Kirikova, Finke et Grundspenkis, 2010). En adoptant un point de vue particulier sur les systèmes d'information, les auteurs ont également proposé leur propre définition en divisant le CIM en deux modèles, *Human Intelligence* et *Artificial Information*. Néanmoins, la structure de ces modèles et les relations de leurs éléments à l'intention d'affaires ne sont pas abordées.

L'objectif de ce chapitre est de développer une telle structure. De notre point de vue, le CIM doit être divisé en une hiérarchie de trois modèles interconnectés : le modèle de la motivation des affaires (BMM), le modèle des processus d'affaires (BPM), et le modèle des exigences (RM). À partir de la définition de l'OMG du CIM, la suite de ce chapitre s'efforce de définir la structure du CIM à partir de ces trois modèles ainsi que les étapes constituant la démarche de sa conception.

Il est important de noter que MDA (Joaquin Miller et Jishnu Mukerji, 2003) ne spécifie aucune méthodologie quant à l'élaboration de ce modèle et la façon de le transformer en le modèle PIM, ce qui rend cette tâche dépendante de l'expérience et de la compétence des analystes et architectes d'affaires.

3.2 Le CIM selon l'OMG

Dans son guide (Joaquin Miller et Jishnu Mukerji, 2003), l'OMG définit le CIM comme suit :

« *A computation independent model is a* ***view of a system from the computation independent viewpoint***. ***A CIM does not show details of the structure of systems***. *A* ***CIM is sometimes called a domain model*** *and a vocabulary that is familiar to the practitioners of the domain in question is used in its specification.* ***It is assumed that the primary user of the CIM, the domain practitioner, is not knowledgeable about the models or artifacts used to realize the functionality for which the requirements are articulated in the CIM***. *The CIM plays an important role in bridging the gap between those that are experts about the domain and its requirements on the one hand, and those that are experts of the design and construction of the artifacts that together satisfy the domain requirements, on the other.*

The requirements for the system is modeled in a computation independent model, CIM describing the situation in which the system will be used. *Such a* ***model is sometimes called a domain model or a business model***. ***It may hide much or all information about the use of automated data processing systems.*** *Typically such a model* ***is independent of how the system is implemented.***

A CIM is ***a model of a system that shows the system in the environment in which it will operate****, and thus it helps in presenting exactly what the system is expected to do. It is useful, not only as an aid to* ***understanding a problem****, but also as a* ***source of a shared vocabulary for use in other models.*** *In an MDA specification of a system* ***CIM requirements should be traceable to the PIM*** *and PSM constructs that implement them, and vice versa.*
A CIM ***might consist of two UML models****, from the* ***ODP enterprise and information viewpoints****. It* ***might include several models from these viewpoints****,* ***some providing more detail than others****, or* ***focusing on particular concerns of a viewpoint.*** *»*

Les énoncés suivants résument cette définition du CIM :

1. Il est indépendant de toute computation;
2. Il ne montre pas les détails de la structure du système;
3. Il est souvent appelé le modèle de domaine ou le modèle d'affaires et utilise un vocabulaire familier aux praticiens du CIM;
4. Il permet de combler le fossé entre les experts du domaine d'affaires et les experts techniques;
5. Il montre le système dans l'environnement auquel il sera installé;
6. Les exigences du système sont modélisées dans le CIM;
7. Ses éléments doivent être traçables aux éléments du PIM.

Cet ensemble d'énoncés constitue un aperçu de ce qu'est un CIM. Toutefois, il ne définit pas précisément la structure des éléments constituants ce niveau de MDA qui, conformément à l'énoncé 7, est nécessaire pour être transformé en PIM. Certains travaux ont été réalisés dans cette direction (par exemple (Osis Janis, Asnina Erika et Grave Andrejs, 2009)), mais le manque de précision empêche une définition consensuelle. Lors de la création du CIM, l'analyste est ainsi laissé à ses expériences et sa créativité.

Afin de standardiser cette activité, nous proposons une structure d'éléments composée de trois modèles (Samir Kherraf et al., 2010) comme présentés dans la Figure 3.1.

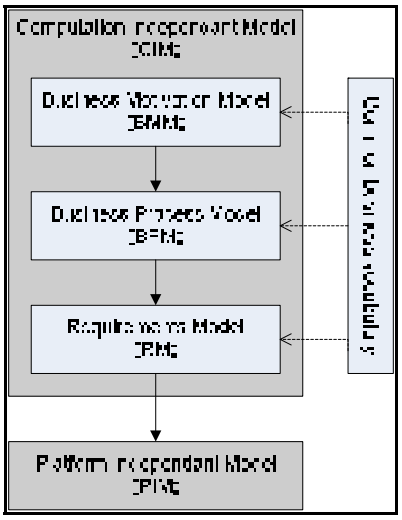


Figure 3.1 Vue d'ensemble de la structure du CIM proposée

Le troisième modèle constituant le CIM est celui des exigences (*Requirements Model* (RM)). Il présente ce que le système doit faire sur le plan conceptuel. Les cas d'utilisation sont généralement employés à cette fin (Bente Anda et Dag I.K. Sjøberg, 2003; Reynaldo Giganto et Tony Smith, 2008; Roussev, 2003; Samir Kherraf, Eric Lefebvre et Witold Suryn, 2008). Les exigences doivent traiter le système comme une « boîte noire » afin de se conformer aux énoncés 1 et 2. L'énoncé 3 suggère de limiter le vocabulaire utilisé dans la rédaction des exigences au domaine d'application, évitant ainsi des détails techniques tels que les concepts de base de données ou la description des interfaces utilisateurs. Il est démontré dans (Samir Kherraf, Eric Lefebvre et Witold Suryn, 2008) et les chapitres 9 et 10 de cette thèse comment transformer le RM en PIM. Cette transformation montre que le RM est conforme aux énoncés 4, 6 et 7.

Le second modèle du CIM est le BPM. Il décrit les opérations qui doivent être prises en charge par le système. Plus précisément, le BPM décrit une séquence d'activités nécessaires pour atteindre un but organisationnel (WFMC, 1999). Ces activités transforment généralement des informations, d'une manière manuelle ou automatique, et sont réalisées par différentes unités organisationnelles. Ainsi, le BPM est conforme aux énoncés 3 et 5.

Le *Business Motivation Model* BMM représente le modèle du plus haut niveau du CIM. Il décrit l'organisation dans laquelle les processus d'affaires sont créés et les objectifs qu'ils visent à atteindre. Le BMM représente un modèle de connaissance du CIM. Il est également, comme l'approche MDA, une spécification de l'OMG (OMG, September 2007). Autrement dit, il s'agit d'un métamodèle des principaux éléments nécessaires pour décrire un plan d'affaires : *end*, *means*, *influencer* and *assessment*. Il complète le BPM en couvrant les énoncés 3 et 5.

Donc, lors de la création du CIM, le BMM représente le premier modèle à réaliser. Il décrit l'ensemble des intentions d'affaires. De cette description, le BPM peut être créé pour supporter les moyens (*means*) et les fins (*ends*) définis dans le BMM. Sur la base du BPM, le RM peut enfin être créé pour soutenir l'automatisation et la transformation du CIM en PIM.

3.3 Définition du CIM : notre proposition

L'évolution technologique et les changements organisationnels fréquents ont un impact majeur sur l'utilisation du système informatique. Comme nous désirons maintenir un haut niveau d'agilité et de qualité, il est important de savoir les identifier et de comprendre leurs impacts. Toutefois, les applications logicielles développées uniquement par la technique des cas d'utilisation ne permettent pas à l'organisation un bon niveau de réactivité face aux changements (Birol Berkem, 2008b; Jim Amsden, 2008): ces systèmes sont structurés en fonction de l'intention de leurs utilisateurs plutôt que d'être structurés suivant les objectifs d'affaires et les règles qui les supportent. En conséquence, ces systèmes ne sont pas en mesure de saisir les changements de décision au niveau managérial tels que les tactiques, les règles d'affaires et l'évolution des responsabilités des utilisateurs du système et de les acheminer d'une façon cohérente vers les applications logicielles. Dans ce contexte, le BMM permet de doter le CIM d'une réflexion conceptuelle favorisant la définition d'un ensemble d'éléments interreliés représentant des concepts-clés du domaine d'affaires comme la vision, la mission, les directives et les stratégies d'affaires. Ce modèle, positionné en amont du BPM

et du RM, permet de fournir la possibilité d'identifier quel cas d'utilisation réalise quelle motivation d'affaires (vision, buts, objectifs, missions, stratégie, politique, etc.).

Nous avons présenté, dans la section précédente, un aperçu de la structure proposée du CIM. Dans cette section, nous décrivons, plus en détails, les relations entre les trois modèles qui composent le CIM, du modèle du plus haut niveau vers celui du plus bas niveau : BMM, BPM et RM.

3.3.1 Business Motivation Model (BMM)

Le BMM offre une structure pour développer, communiquer et gérer des plans d'affaires d'une manière organisée. Les éléments qui le constituent sont de méthodologie dite « neutre ». Il n'existe pas une méthodologie particulière permettant la conception du BMM, lui offrant ainsi une certaine flexibilité et agilité. Le BMM se base sur un vocabulaire d'affaires qui est souvent sous forme de langage naturel. Il sert de premier modèle de référence pour définir les éléments des deux autres modèles du CIM, le modèle des processus d'affaires et le modèle des cas d'utilisation.

Le BMM se décompose en deux grandes zones qui définissent deux dimensions majeures : la première concerne les Moyens et les Fins, et la seconde les influenceurs et les évaluations.

La Figure 3.2 présente le BMM selon (OMG, September 2007). Il se compose de quatre éléments principaux : *End*, *Means*, *Influencer* et *Assessment*.

End que nous pouvons traduire par « Fin » ou « Finalité », décrit ce qu'une organisation désire devenir et cherche à accomplir. Par exemple, elle vise à devenir le leader dans son domaine ou à réduire ses risques financiers. *End* peut être une *Vision* (vision) ou un *Desired Result* (résultat désiré) qui, à son tour, peut être soit un *Goal* (but), soit un *Objective* (objectif). Une *Vision* est facultative. Elle se contente de donner un aperçu de *End*. Pour sa part, *Desired Result*, de type *Goal* ou *Objective*, est plus spécifique. Un *Objective* est une

étape dans la réalisation du *Goal*. Il peut être utilisé pour mesurer le progrès vers la réalisation du *Goal*. Bien que la réalisation d'un *Goal* se concrétise habituellement à long terme et peut être définie qualitativement, un *Objective* a la particularité d'avoir une date de fin. En effet, il doit être défini de manière à savoir s'il a bien été réalisé ou non.

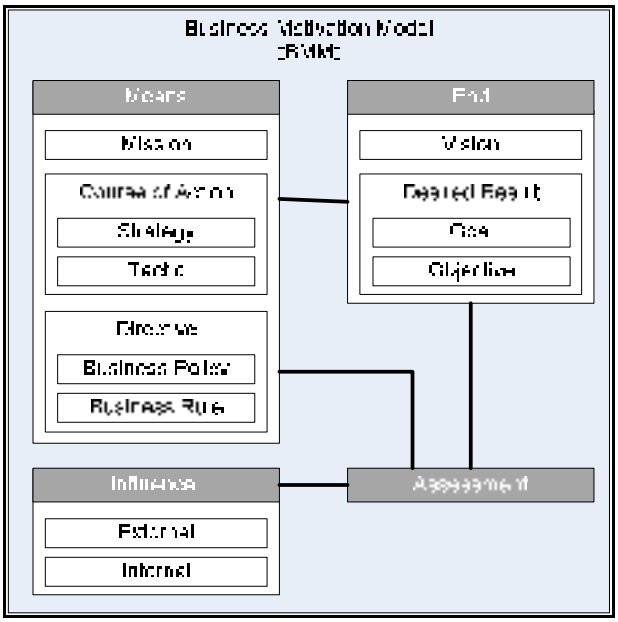


Figure 3.2 La structure du BMM
Empruntée de (OMG, September 2007).

Les ***Means***, que nous pouvons traduire par « moyens », décrivent comment les organisations décident de parvenir à une *End*. Les *Means* peuvent être une *Mission* (mission), un *Course of Action* (plan d'action) ou une *Directive* (directive). Bien que facultatives, les *Means* décrivent d'une manière large l'activité d'une organisation. Le *Course of Action* définit ce qui doit être fait sans préciser comment le faire. Il peut s'agir d'une *Strategy* (stratégie) ou d'une *Tactic* (tactique). Le *Course of Action* est lié directement au *Desired Result*. Il est suggéré que la *Strategy* soit liée à la réalisation des *Goals*, et la *Tactic* à la réalisation des *Objectives*. Une *Directive* (directive), qui peut être soit une *Business Policy* (politique d'affaires) ou une *Business Rule* (règle d'affaires), régit le *Course of Action*. Les *Business Policies* fixent les limites de ce qui peut être fait et de ce qui ne peut pas l'être. Les *Business Rules* doivent être actionnables et issues des *Business Policies*.

Influencer, que nous pouvons traduire par « influenceur », affecte les *Means* et les *Ends*. Il peut être à l'interne ou à l'externe de l'organisation. Les *Influencers* externes peuvent être de différents types tels que les *Competitors* (concurrents), *Customers* (clients), *Environments* (environnements), *Partners* (partenaires), *Regulations* (règlements) ou *Technology* (technologie). Les *Influencers* internes peuvent être également de différents types tels que *Corporate Value* (valeur corporative), *Habits* (habitudes) ou *Infrastructure* (infrastructure).

Assessment, que nous pouvons traduire par « évaluation », est un jugement de certains *Influencers* qui affectent la capacité de l'organisation d'utiliser ses *Means* ou atteindre ses *Ends*. En d'autres termes, un *Assessment* exprime une connexion logique entre les *Influencers* et les *Means* et/ou les *Ends* du plan d'affaires. De cette façon, un *Assessment* indique qui sont les *Influencers* pertinents pour tels *Ends* et/ou pour tels *Means*. Les *Influencers* sont de nature neutre jusqu'à ce qu'une évaluation soit faite à leur égard.

3.3.2 Business Process Model (BPM)

Dans les spécifications du BMM (OMG, September 2007), l'OMG a relié les éléments du BMM aux éléments externes suivants : *Business Process* (processus d'affaires), *Organization Unit* (unité organisationnelle) et *Business Rules* (règles d'affaires). Ces trois éléments composent le BPM qui est défini plus en détails en (OMG, 2008). La Figure 3.3 présente le BPM et sa relation avec le BMM.

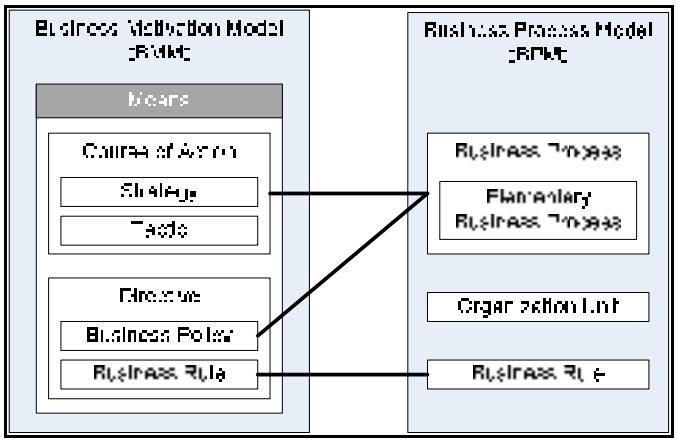


Figure 3.3 Structure du BPM et sa relation avec le BMM

Business Process réalise le *Course of Action*, principalement en terme de séquences d'étapes. Ceci justifie sa relation avec le BMM. Un *Business Process* est régi par les *Business Policies*. Il peut être décomposé en processus d'affaires élémentaire (Elementary Business Process (EBP)) défini dans (Larman, 2004) comme « *a task performed by one person in one place at one time, in response to a business event, which adds measurable business value and leaves the data in a consistent state (...)* ». Chaque EBP doit être sous la responsabilité d'une unité organisationnelle (propriétaire du processus). La modélisation des processus d'affaires est largement discutée dans la littérature. Le « Workflow Management Coalition » (WFMC, 1999) définit un processus d'affaires comme « *a set of one or more linked procedures or activities which collectively realize a business objective or policy goal, normally within the context of an organizational structure defining functional roles and relationships* ».

Les langages les plus utilisés sont UML avec son diagramme d'activités et le BPMN (OMG, 2007). MDA préconise UML comme standard de modélisation. Dans ce sens, le diagramme d'activités est également recommandé convenable pour la modélisation des processus d'affaires (Grady Booch, James Rumbaugh et Ivar Jacobson, 2005; Marlon Dumas et Arthur H. M. Ter Hofstede, 2001). Dans notre recherche, celui-ci sera adopté pour modéliser les processus d'affaires.

Organization Unit : dans le BPM, les processus d'affaires sont toujours sous la responsabilité d'une unité organisationnelle. Celle-ci peut représenter n'importe quel élément qui structure une organisation à l'interne ou à l'externe comme un partenaire d'affaires, un département, une équipe ou un rôle joué par une partie prenante. Une unité organisationnelle définit les *Ends*, établit les *Means*, reconnaît les *Influencers*, permet les *Assessments*, est définie par des *Strategies* et est responsable des *Business Processes*.

Les ***Business Rules***, ou règles d'affaires du BPM, proviennent directement de celles du BMM. Elles servent à orienter les processus d'affaires. Selon (OMG, September 2007), elles fournissent les bases des décisions qui doivent être prises au sein d'un processus d'affaires. Une règle d'affaires est une *Directive* en soi destinée à gouverner, guider ou influencer le comportement des affaires. Elle appuie la *Business Policy* qui a été formulée en réponse à une opportunité, une menace, une force ou une faiblesse. C'est une *Directive* unique qui n'a pas besoin d'être interprétée pour entreprendre les *Strategies* ou les *Tactics*. Elle est souvent dérivée de *Business Policies* et guide les *Business Processes*.

Formellement, une règle d'affaires est une règle sous la juridiction d'affaires. Elle introduit toujours une obligation ou une nécessité.

3.3.3 Requirement Model (RM)

Le développement de logiciel est basé sur la définition des exigences. La Figure 3.4 illustre le RM et sa relation avec le BPM. Nous considérons, dans cette recherche, que le RM est composé de trois éléments : cas d'utilisation, acteur et des règles d'affaires.

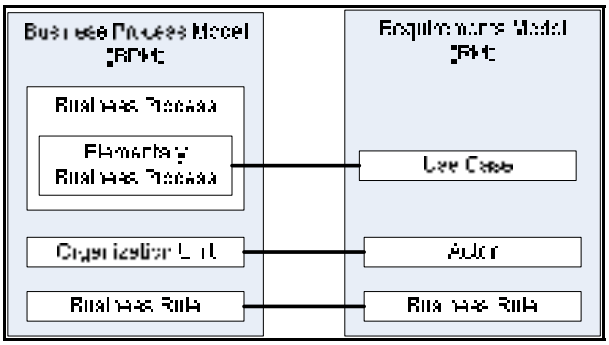


Figure 3.4 Structure du RM et sa relation avec le BPM

Le ***Use Case***, ou cas d'utilisation, décrit les interactions entre un acteur et le système à travers une série d'étapes appelée scénario. Il existe donc un scénario principal qui décrit l'enchaînement nominal des actions de l'acteur et du système. Dans la plupart des cas, il y a des scénarios alternatifs qui décrivent les cas d'exceptions et les cas d'erreurs.

Un cas d'utilisation décrit les détails d'un EBP. Ainsi, le RM définit uniquement la structure des relations entre les cas d'utilisation, à l'aide du modèle de cas d'utilisation, par le biais des deux associations *include* et *extend*. La transformation d'un EBP en un cas d'utilisation a été démontrée dans (Samir Kherraf, Eric Lefebvre et Witold Suryn, 2008) et sera détaillée dans le chapitre 7 de cette thèse.

Actor ou acteur est une entité physique ou logique, qui bénéficie du cas d'utilisation. Un acteur physique est une personne comme un directeur ou un client et, un acteur logique peut être un groupe de personnes comme un ministère ou un système comme par exemple le système bancaire. Un cas d'utilisation a besoin d'un acteur primaire et, dans certains cas, d'un ou plusieurs acteurs secondaires. Ceux-ci ne bénéficient pas du cas d'utilisation, mais sont indispensables pour sa réalisation. Par exemple, dans un cas d'utilisation qui porte sur un paiement par carte de crédit, le client est l'acteur primaire. Comme le système doit échanger avec le système bancaire pour valider le paiement, le système bancaire est considéré l'acteur secondaire. Un acteur, primaire ou secondaire, est lié directement à une unité organisationnelle dans le BPM.

Les ***Business Rules***, ou règles d'affaires du RM, sont liées aux règles d'affaires du BPM. Elles sont généralement responsables du déclenchement de scénarios alternatifs dans le BPM et les cas d'utilisation.

3.4 Vers un modèle générique du CIM

Le BMM offre une connexion logique entre les différents éléments qui le composent. Cette connexion permet d'instaurer une traçabilité entre les éléments d'affaires et les unités organisationnelles dont elle est responsable. Dans le but de proposer un modèle générique du CIM, nous avons retenu le modèle BMM, le modèle des processus d'affaires et le modèle de cas d'utilisation.

Il est à noter, comme le montre la Figure 3.5, que le BMM et le modèle des processus d'affaires sont empruntés des spécifications de l'OMG. Les deux éléments *Role* et *Elementary Business Process* ont été ajoutés au BPM pour consolider notre approche de sa conception, qui sera détaillée dans le chapitre 5.

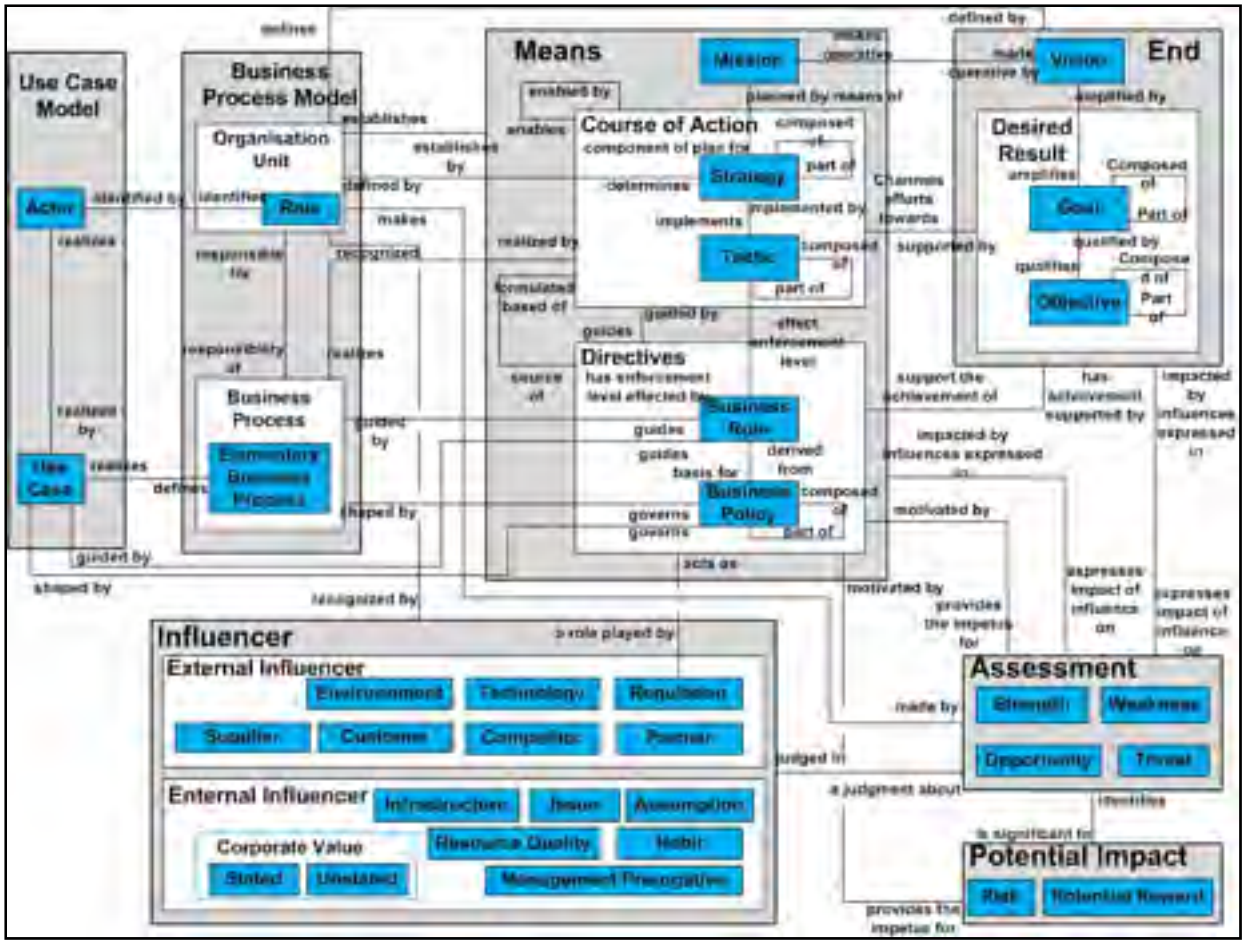


Figure 3.5 Proposition de l'architecture du modèle générique du CIM

3.5 Vers une méthodologie de construction du CIM

Le BMM est neutre à l'égard de toute méthodologie. L'OMG préconise son utilisation comme un plan d'affaires pouvant supporter une variété de méthodologies : sa mise en œuvre se traduirait par un référentiel dans lequel ses éléments pourraient être emmagasinés, connectés, et liés à d'autres informations de l'organisation. L'OMG a proposé (OMG, September 2007) un exemple de méthodologie à suivre (ellipses noires dans la Figure 3.6) que nous avons retenu et avons complété (ellipses bleues dans la Figure 3.6) pour généraliser cette méthodologie afin de construire le CIM.

La Figure 3.6 illustre notre approche méthodologique de construction du CIM. Les ellipses représentent les activités à réaliser et les flèches les dépendances logiques entre les activités.

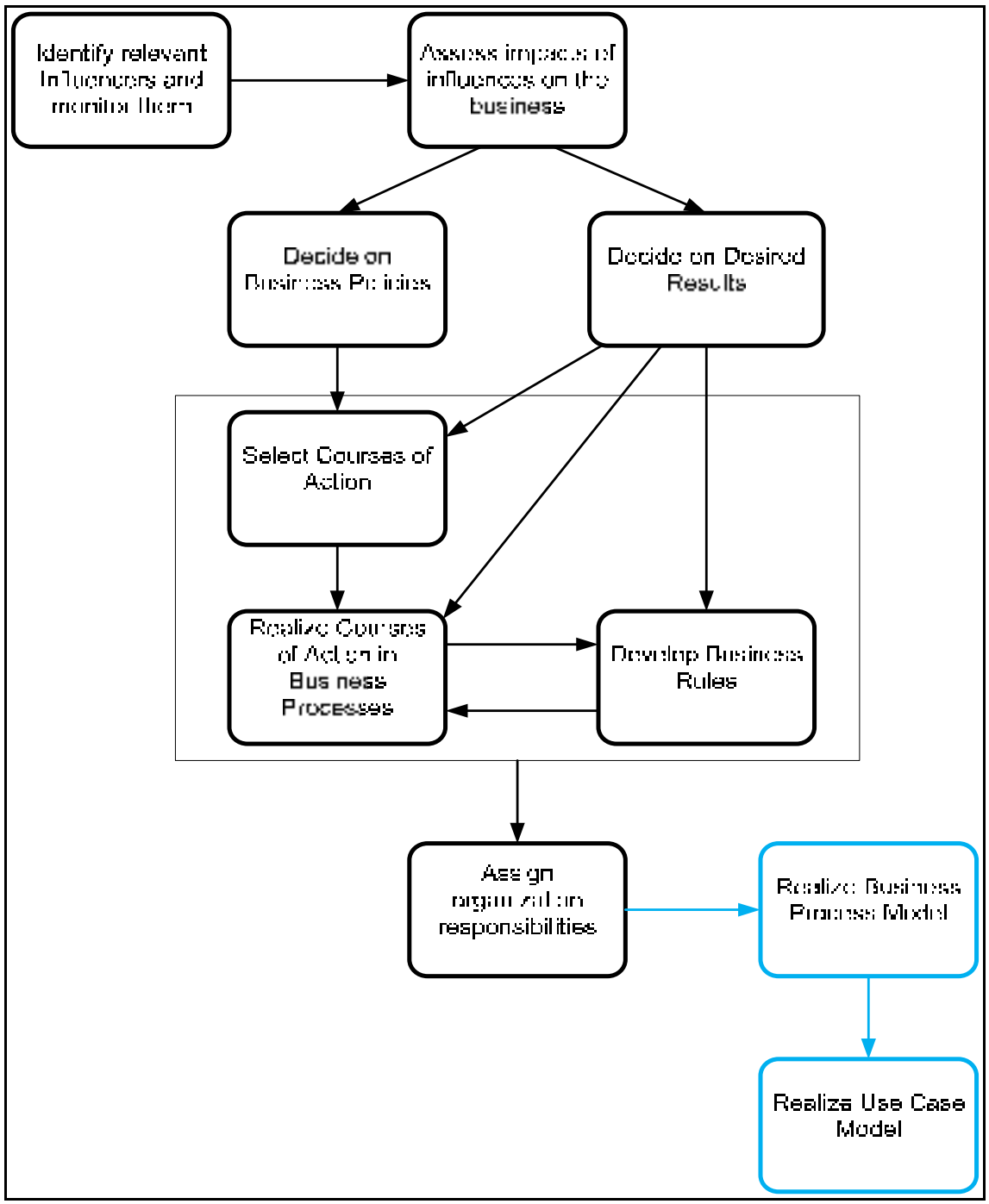


Figure 3.6 Proposition d'une méthodologie de construction du CIM

3.6 Discussion

Afin de faciliter la création du CIM, il est important de définir une méthodologie et un processus de transformation permettant le passage du BMM au BPM et du BPM au RM. Nous avons proposé une méthodologie sommaire de construction du CIM et un processus muni d'un ensemble de règles visant à transformer un BPM en un RM dans (Samir Kherraf, Eric Lefebvre et Witold Suryn, 2008). Des améliorations de ces règles et de leur mise en œuvre par un cas d'étude seront abordées dans les chapitres 5, 7 et 8 de cette thèse.

L'utilisation du BMM offre un moyen simple de gérer, avec un haut niveau d'abstraction, les exigences d'affaires. Il est suffisamment générique pour spécifier les exigences de n'importe quel domaine d'affaires. Un outil logiciel permettra d'améliorer la compréhension des relations entre l'ensemble des éléments constituant le CIM.

Afin de bâtir notre stratégie de transformation du BMM au BPM et du BPM au RM, il est essentiel de définir un profil UML pour chacun d'eux. Ces profils, qui doivent être conforme au MOF, faciliteront la transformation et la traçabilité entre l'ensemble des éléments du CIM.

Dans cette thèse, nous nous intéressons uniquement aux deux modèles BPM et RM. Le BMM sera l'objet d'une recherche future.

CHAPITRE 4

CONCEPTION DE L'ARCHITECTURE DU <<PLATFORM INDEPENDENT MODEL>> (PIM)

4.1 Introduction

Ce chapitre présente la phase 3 de ce projet de recherche, soit la définition de l'architecture du PIM et sa méthode de construction.

Le PIM représente le deuxième modèle en importance dans l'approche MDA. Il doit être indépendant des plateformes d'exécution et d'implémentation. Cela veut dire que dans un PIM, nous décrivons le fonctionnement des entités et des services sans montrer les détails de son utilisation sur une plateforme technique particulière.

Le PIM doit garder un certain degré d'indépendance vis-à-vis des plateformes afin d'approprier son utilisation à différents types de plateformes. D'après l'OMG (Joaquin Miller et Jishnu Mukerji, 2003), le PIM représente le modèle de classe d'analyse, et sa conception doit respecter ainsi que prendre en compte l'ensemble des éléments constituant le CIM. Théoriquement, il doit traduire les exigences logicielles en un format générique et exploitable par différentes plateformes. Pour construire le PIM, l'approche MDA recommande l'utilisation du formalisme UML. De ce point de vue, et comme démontré dans plusieurs travaux antérieurs (Eric Lefebvre, 2005; Janis Osis et Uldis Donins, 2010; Samir Kherraf, Eric Lefebvre et Witold Suryn, 2008), le diagramme de classe représente le meilleur moyen de construire le PIM.

4.2 Conception du PIM

Nous avons proposé dans le chapitre précédent l'ensemble des éléments constituant la structure du CIM. Nous sommes conscients que l'objectif principal de cette recherche est de

concevoir une méthodologie qui permette la transformation du CIM vers le PIM. Certains travaux se sont focalisés sur ce mécanisme de transformation sans définir clairement la structure, les frontières et la manière de conception de chacun des deux modèles.

Afin de proposer une structure générique du PIM ainsi que le processus permettant la transformation du CIM au PIM, nous devons examiner de nouvelles méthodes et techniques. Pour cela, nous allons d'abord explorer les patrons d'affaires et les patrons d'exigence. Ceux-ci font référence aux processus d'affaires et aux cas d'utilisation, deux modèles principaux et essentiels dans la conception du CIM. Ensuite, nous étudierons les patrons d'analyse qui font référence au modèle de classe d'analyse que nous considérons comme modèle principal dans la conception du PIM.

Cette façon de faire nous permettra d'explorer si les patrons d'affaires, d'exigence et d'analyse peuvent guider le processus de découverte de liens entre les éléments constituant respectivement le CIM et le PIM. Ces liens sont pour nous d'une importance cruciale pour établir une traçabilité entre le CIM et le PIM. Celle-ci représente le point de départ pour automatiser la transformation du CIM au PIM.

Pour ce faire, nous rappellerons d'abord le *Rational Unified Process* (RUP), un processus de développement logiciel largement utilisé, tout en mettant le lien avec l'approche MDA notamment le CIM et le PIM et leur équivalent dans le RUP. Nous examinerons ensuite les patrons d'affaires, les patrons des exigences, les patrons d'analyse existants, les archétypes et le *Domain Neutral Component* (DNC). Par la suite, nous proposerons une méthode de construction du PIM. Enfin, nous expérimenterons, dans ce chapitre, notre méthode par un cas d'étude.

4.3 Technologies reliées à la conception du PIM

4.3.1 Rational Unified Process (RUP)

Le RUP (Kruchten Philippe, 2000) est un cadre de développement logiciel d'une manière itérative. Il se propose comme un guide d'utilisation efficace d'UML et préconise six bonnes pratiques de développement de logiciels. Il définit un ensemble de neuf disciplines regroupant les activités du cycle de développement logiciel. Nous nous intéressons aux trois premières qui concernent la modélisation, les exigences d'affaires ainsi que l'analyse et la conception de système. En effet, ces trois disciplines font partie intégrante des modèles CIM et PIM. Elles sont :

- La modélisation d'affaires (*Business modelling*)

Cette discipline consiste à capturer, modéliser et analyser les processus d'affaires ainsi que les rôles et les responsabilités de chaque intervenant dans le domaine de problème. Les principaux artefacts sont le modèle de processus d'affaires et le modèle d'objet d'affaires. Le premier se base sur les cas d'utilisation d'affaires (business use case), représentant les objets d'affaires (au niveau conceptuel et non au niveau système) et le deuxième détaille davantage les services, les parties prenantes et les objets utilisés.

- Les exigences (*Requirements*)

Cette discipline définit le document de vision du système et identifie successivement les besoins des utilisateurs, les fonctionnalités du système et les exigences logicielles. Le modèle des cas d'utilisation représente l'artefact principal de cette discipline. Les cas d'utilisation sont décrits en détails et les exigences non fonctionnelles sont décrites comme des spécifications supplémentaires.

- L'analyse et la conception (*Analysis and Design*)

L'objectif de cette discipline réside dans la traduction des exigences logicielles, telles qu'exprimées dans le modèle des processus d'affaires et le modèle des cas d'utilisation, dans une représentation du système destinée aux développeurs. Les principaux artefacts sont :

- Le modèle de classe d'analyse qui représente les objets d'affaires du domaine de problème;
- Le modèle de classe de conception qui ajoute au modèle de classe d'analyse des objets système ainsi que d'autres détails techniques selon la plateforme d'exécution et le langage de programmation utilisé.

4.3.2 RUP et MDA

Bien que la modélisation des processus d'affaires puisse se faire de manière indépendante ou dans le cadre de la réingénierie des processus, elle ne fait qu'ajouter de la valeur au cycle de développement logiciel. Plus précisément, selon le RUP, la modélisation d'affaires contribue à définir les exigences systèmes.

En fait, un modèle d'affaires basé sur UML peut être une entrée directe vers un modèle d'exigences. RUP (John, 2003) définit la manière de transformer un modèle de processus d'affaires en un modèle de classe d'analyse comme suit :

- Un processus d'affaires (cas d'utilisation d'affaires dans le jargon RUP) qui doit être automatisé sera représenté par un cas d'utilisation. Il deviendra un sous-système (une fonction du système);
- Chaque entité d'affaires (objet d'affaires) se transforme en une classe dans le modèle de classe d'analyse.

Cette transformation nous est utile car elle s'aligne avec notre objectif, celui de transformer le CIM en PIM. En effet, elle consolide la validité de notre proposition des modèles constituants le CIM et le PIM : le modèle des processus d'affaires et le modèle des cas d'utilisation font partie du CIM et, le modèle de classe d'analyse fait partie du PIM.

4.3.3 Patrons d'affaires et patrons d'analyse

Les patrons identifient des solutions réutilisables aux problèmes récurrents. Le *Gang of Four design patterns* (Erich Gamma et al., 1994) sont les patrons les plus utilisés : ils représentent un ensemble de 23 patrons déduits des bonnes pratiques de conception de logiciels. Ils constituent ainsi une référence dans de nombreux cas et ils ont été adaptés à plusieurs architectures logicielles modernes. Les patrons GRASP (General Responsibility Assignment Software Patterns) (Larman, 2004) sont aussi largement utilisés : ils décrivent des règles pour affecter les responsabilités aux classes de conception en liaison avec la méthode de conception *Boundary Control Entity* (BCE).

L'utilisation des patrons semble faciliter grandement la conception de logiciel et la communication entre les architectes. Ils sont essentiels à l'approche MDA dans l'étape de génération d'un PSM à partir d'un PIM mais n'aident pas à concevoir le modèle d'analyse et le modèle d'affaires (Eric Lefebvre, 2005).

En effet, il existe une initiative (Wil van der Aalst et Arthur ter Hofstede, 1999) appelée *workflow patterns* initiée en 1999 qui s'intéresse à fournir une base conceptuelle à la technologie des processus d'affaires à différents points de vue (flux de contrôle, données, ressources et la gestion des exceptions). Ces patrons servent à évaluer la pertinence, les forces, les faiblesses et la compatibilité d'un langage de modélisation des processus d'affaires pour la réalisation d'un projet particulier.

Les patrons pouvant s'appliquer aux disciplines de RUP, les exigences logicielles (cas d'utilisation) et la phase de l'analyse sont encore rares et leurs utilisations sont limitées.

4.3.3.1 Les patrons de la modélisation d'affaires

Un processus d'affaires est une séquence ordonnée et chronologique d'activités destinées à consommer des ressources et à produire un résultat. La modélisation des processus d'affaires

vise donc à comprendre et analyser l'ensemble des activités du domaine de problème avant le développement du système informatique qui soutiendra le système d'information.

Un patron de processus d'affaires offre une démarche qui permet d'identifier les activités, les éléments entrants et les éléments sortants afin d'aboutir au modèle de processus d'affaires.

À la fin des années 70, IBM a proposé une méthode, *Business Service Planification* (BSP) (David V. Kerner, 1979), utilisée dans un premier temps à l'interne puis à l'externe pour l'ensemble des clients IBM. Cette méthode couvre le cycle de développement d'une application logicielle, de la phase de planification jusqu'à la phase d'implémentation. Elle met l'accent sur les processus d'affaires issus de la mission, des objectifs et des buts de l'organisation. Ces processus d'affaires sont analysés pour en extraire les données et les classes de données pour ensuite développer la base de données. Le plan BSP final décrit l'architecture de l'entreprise.
Cette méthode utilise des patrons qui se réfèrent au cycle de vie d'une ressource (humaine, matérielle ou financière) tout en identifiant des processus d'affaires génériques (*Plan, Acquire, Use, Evaluate, Maintain* et *Dispose*). Ceux-ci peuvent être décomposés à nouveau en sous-processus pour en créer d'autres à chaque fois. Ils sont réutilisables et configurables à chaque fois qu'il y a un besoin de gérer une ressource.

Durant la même période, IBM a créé un autre processus de planification nommé *Business Information Characterization Study* (BICS). Ce processus est une extension logique de la méthodologie BSP et utilise la théorie de *Business Information Analysis and Integration Technique* (BIAIT) dans le but de fournir une approche structurée pour comprendre les systèmes d'information d'affaires. Cette démarche a permis d'identifier un ensemble d'environ 75 fonctions génériques et 50 entités d'affaires (business entity) génériques adaptées à des contextes spécifiques (David V. Kerner, 1979; Walter M. Carlson, 1979) pour ainsi construire rapidement une architecture du système d'information.

En se basant sur les modèles BICS et BSP, Lefebvre (Lefebvre Éric, 1996) a proposé un patron qui caractérise une activité humaine définie comme suit :

Toute activité humaine :

- a un objectif,
- est déclenchée par une activité précédente,
- a une personne responsable,
- requiert, pour sa réalisation, des ressources :
  - humaines,
  - matérielles (consommables),
  - d'infrastructure (actifs),
  - financières,
- a un résultat selon l'objectif.

Ce patron sert à définir un processus d'affaires élémentaire (Elementary Business Process (EBP)) (Eric Lefebvre, 2005; Samir Kherraf, Eric Lefebvre et Witold Suryn, 2008), c'est-à-dire un processus d'affaires que nous ne sommes pas intéressés à décomposer davantage. Il correspond à une activité, bien définie et bien délimitée, d'une partie prenante. La Figure 4.1 présente une illustration de l'EBP, basée sur la chronologie des activités et les entités d'affaires requises par l'activité. Les entités d'affaires englobent respectivement les ressources utilisées et les produits/services créés par l'activité.

Dans ce schéma (Figure 4.1) le *InputEvent* fait référence à l'activité précédente et le *OutputEvent* fait référence à l'activité suivante.

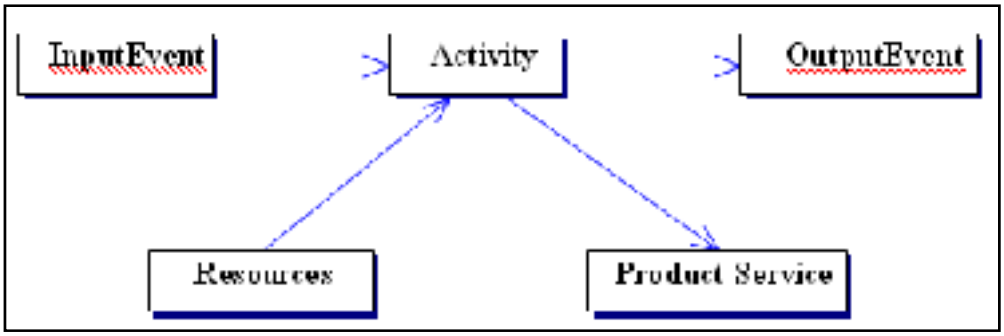


Figure 4.1 Le modèle du processus d'affaires élémentaire

4.3.3.2 Les patrons des exigences

Peu de travaux ont été établis pour identifier les patrons des exigences, notamment dans la spécification et l'écriture des cas d'utilisation. Les travaux de Stephen Withall (Withall, 2007) comptent parmi les plus importants mais ne traitent pas les cas d'utilisation. Nous avons retenu l'initiative la plus importante pour notre recherche (Eric Lefebvre, 2005; Samir Kherraf, Eric Lefebvre et Witold Suryn, 2008), celle d'utiliser l'EBP, qui sera détaillée dans le chapitre 5, comme point de départ pour spécifier les exigences dans un modèle de cas d'utilisation.

4.3.3.3 Les patrons d'analyse

À la différence des patrons de conception, les patrons d'analyse capturent les aspects conceptuels (organisationnels, affaires, sociaux, etc.) dans un domaine de problème. Ils sont utiles, à la fois pour accélérer le développement logiciel en fournissant des modèles d'analyse réutilisables et pour faciliter la transformation du modèle d'analyse en modèle de conception.

Le terme « patron d'analyse » a été proposé par Martin Fowler (Fowler Martin, 1996) à partir de ses observations lors de projets de développement logiciel. Il en a défini plusieurs portant sur des domaines d'affaires spécifiques comme la comptabilité, mais ces patrons ne permettent pas d'agir comme solution universelle à la construction de tout modèle d'analyse.

Dans le même ordre d'idées, Jacobson a défini trois catégories d'objets (Ivar Jacobson et al, 1992) :

- *Boundary* : pour représenter les interfaces utilisateurs,
- *Control* : un objet qui contrôle l'exécution d'une transaction,
- *Entity* : tout objet utilisé par une transaction.

Taylor a introduit le concept de l'ingénierie de la convergence (Convergent Engineering) qui vise à combiner les problématiques d'affaires et le génie logiciel dans une seule discipline (Taylor David, 1995). Il préconise de focaliser sur la modélisation et la réingénierie des processus d'affaires, comme partie intégrante du cycle de développement logiciel, dans le but de rester flexible dans leurs environnements d'affaires qui subit des changements et des évolutions rapides. Par conséquent, il a proposé un cadre de modélisation de processus d'affaires en utilisant trois éléments génériques :

- *Organisation* : groupement de personnes et d'autres ressources chargées d'exécuter des processus pour atteindre des objectifs;
- *Processus* : une série d'activités orientées vers un but qui consomment et génèrent des ressources;
- *Ressource* : actif de l'organisation consommé ou généré par un processus.

Coad a introduit (Peter Coad, Eric Lefebvre et Jeff De Luca, 1999), en utilisant des concepts similaires à ceux de Taylor et Jacobson, la notion de l'archétype. Il en a proposé quatre :

- *Moment-Interval* : représente un événement/activité qui peut être suivi dans le temps, quelque chose (objet) qui se produit à un moment défini ou sur un intervalle de temps. Il vient souvent avec ses détails (par exemple une commande et la ligne de commande);
- *Role* : représente la façon de participer d'une entité d'affaire, catégorisée comme *Party*, *Place*, *Thing* (PPT), dans l'archétype *Moment-Interval*.
- *Party, Place, Thing (PPT)* : représente les objets qui peuvent jouer un rôle dans les différentes activités (Moment-Interval) du système.
- *Description* : représente une description (catalogue) individuelle d'un des objets PPT.

Ces quatre archétypes ont été identifiés à partir des expériences de Coad et testés dans de nombreux cas. Ils ont des attributs et des méthodes génériques, ils s'interconnectent entre eux

de manière à former un patron permettant de concevoir rapidement et efficacement le modèle de classe d'analyse. Afin de faciliter la construction du modèle de classe, une couleur et un stéréotype ont été attribués à chaque archétype. Ceci permet de rajouter une nouvelle dimension au modèle de classe d'analyse et représente pour notre recherche une piste à examiner pour concevoir le PIM. En somme, nous analyserons en détail chacun des quatre archétypes.

4.4 La modélisation en couleur à l'aide des archétypes

Durant le processus de développement des applications logicielles de gestion, l'expérience a démontré, mis à part le domaine d'affaires, l'apparition des mêmes types de classes d'analyse (Peter Coad, Eric Lefebvre et Jeff De Luca, 1999) qui se retrouvent d'un projet à l'autre. Coad en a identifié quatre en les qualifiant d'archétypes : " *An archetype is a form from which all classes of the same kind more or less follow – including attributes, links, methods, plug-in points, and interactions*". À titre d'exemple, nous citons deux types de ces quatre archétypes :

- des classes qui représentent des transactions d'affaires ou des interactions de différents types comme les réservations, les locations, les commandes, les livraisons, les soumissions, etc.;
- des classes qui représentent des parties prenantes, des choses et des endroits qui participent à ces classes de transactions d'affaires : 1) des parties prenantes comme les personnes et les organisations qui effectuent des commandes, des achats et des réservations, etc. 2) des choses comme les produits et services vendus ou achetés. 3) des endroits comme des magasins et des bureaux où se passent toutes ces affaires, etc.

Toutefois, cette identification de ces types de classe ne se limite pas seulement aux applications logicielles de gestion mais peut s'étendre, bien au-delà, comme les applications logicielles industrielles. Nous citons, à titre d'exemple, des classes qui modélisent des événements (périodes de temps d'une mesure), des parties prenantes (opérateurs), des endroits (lignes de production), et des choses (appareils de mesure).

4.4.1 L'archétype Moment-Interval

L'archétype Moment-Interval (Figure 4.2) représente un moment dans le temps ou un intervalle de temps. Il modélise un élément dont l'utilité réside dans la définition d'une transaction d'affaires. À titre d'exemple :

- Une réservation se produit sur un intervalle de temps, depuis le moment de la réservation jusqu'au moment de son utilisation, annulation ou expiration;
- Une vente se produit à un moment dans le temps soit la date et l'heure de cette vente.

L'archétype *Moment-Interval* peut être identifié facilement dans la conception du diagramme de classe d'analyse. Il représente souvent des transactions d'affaires et des événements importants dont nous devons mémoriser la date exacte. Son identification ne se limite pas seulement aux applications logicielles de gestion mais s'applique aussi aux applications logicielles industrielles comme dans les cas des intervalles de lecture de température ou de pression, de lecture de capteurs, etc. Coad préconise d'identifier les *Moment-Intervals* à partir des activités du modèle de processus d'affaires.

Dans la plupart des travaux en génie logiciel, il est recommandé d'identifier les classes à partir des noms d'objets parus dans les expressions (dans les détails textuels des cas d'utilisation) lors de la phase de rédaction des cas d'utilisation. Or, dans le cas de la découverte des classes *Moment-Interval*, cette technique doit tenir compte, en plus des noms d'objets, des verbes. En fait, les noms qualifiant *Moments-Interval* ont souvent une forme verbale qui fait référence à une activité d'un processus d'affaires. Par exemple, les déclarations de type «Le client réserve une voiture» ou «Le commis ne doit pas louer des voitures en promotion», utilisées dans l'activité de capture des exigences, pourraient manquer l'identification des *Moment-Interval*. Cependant, les *Moment-Intervals* apparaissent dans des expressions de type « La réservation ne doit pas être faite une semaine avant la location » ou « Une facture doit être établie pour chaque location ». Coad (Peter Coad, Eric Lefebvre et Jeff De Luca, 1999) préconise l'identification des *Moment-Intervals* à partir des

activités (processus d'affaires élémentaire) du modèle de processus d'affaires. Nous retenons ce dernier point dans notre démarche d'identification de l'archétype *Moment-Interval*.

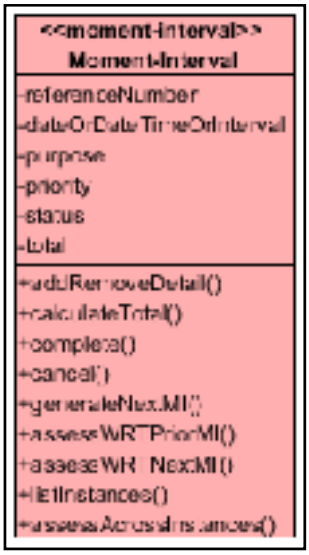


Figure 4.2 L'archétype Moment-Interval
Empruntée de (Peter Coad, Eric Lefebvre et Jeff De Luca, 1999)

4.4.1.1 Relation entre les Moment-Intervals

Généralement, une activité d'un processus d'affaires se transforme en une classe *Moment-Interval* dans le modèle de classe d'analyse (Eric Lefebvre, 2005; Samir Kherraf, Eric Lefebvre et Witold Suryn, 2008). Selon cette règle, nous pouvons organiser les *Moment-Intervals* en une séquence linéaire ou d'ordre complexe, selon l'ordre d'apparition des activités du processus d'affaires.

Les classes *Moment-Interval*s qui se produisent avant ou après un *Moment-Interval* particulier sont nommées respectivement *prior-MI* et *next-MI* (Figure 4.3).

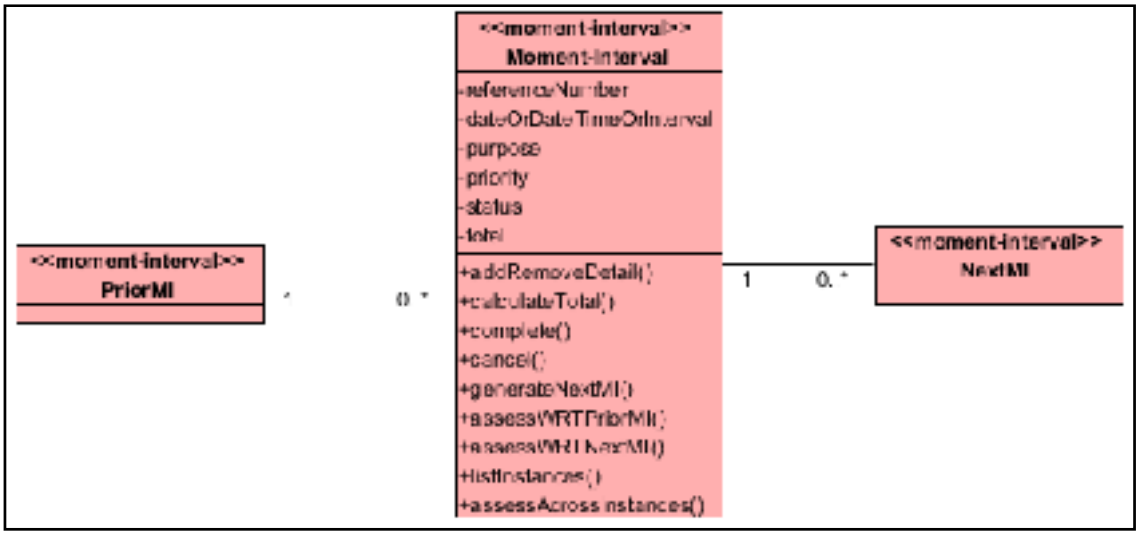


Figure 4.3 Relation entre les archétypes Moment-Intervals
Emprutée de (Peter Coad, Eric Lefebvre et Jeff De Luca, 1999)

4.4.1.2 MI-detail

Une classe *Moment-Interval* a souvent besoin d'une classe qui exprime ses détails. Il peut y avoir plusieurs items de différents types que le *Moment-Interval* a besoin de sauvegarder ou de manipuler. Par exemple, une commande est susceptible de comporter plusieurs types de produits avec des quantités différentes. Par conséquent, la classe commande doit être liée à une classe DétailCommande par une relation d'agrégation du côté de *Moment-Interval* (Figure 4.4). La classe DétailCommande, appelée *MI-Detail*, sert à mémoriser des informations sur chaque type de produit impliqué dans la classe *Commande*.

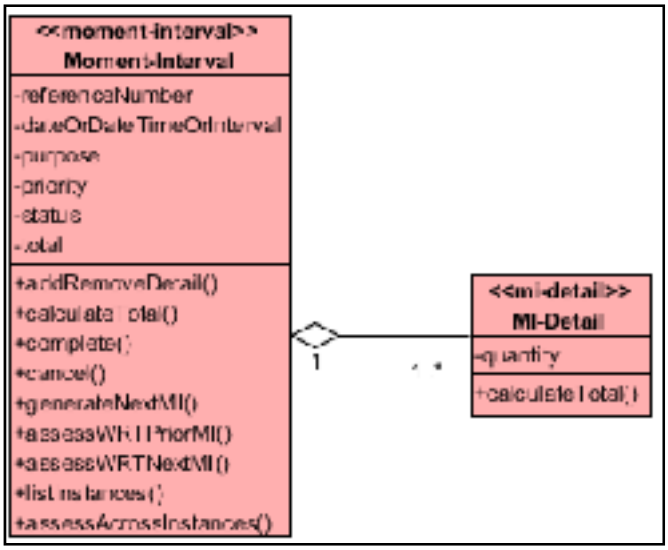


Figure 4.4 L'archétype Moment-Interval et son détail Mi-Detail
Empruntée de (Peter Coad, Eric Lefebvre et Jeff De Luca, 1999)

4.4.1.3 Les responsabilités du Moment-Interval

L'utilité des classes archétypes dans la conception du diagramme de classe réside dans leurs responsabilités typiques. Celles-ci se démarquent par des associations entre les classes archétypes ainsi que des attributs et des opérations spécifiques à chacune d'elles. L'archétype *Moment-Interval* est considéré comme l'archétype le plus important dans la conception du diagramme d'analyse. Il contient les attributs et les méthodes les plus importantes. Le Tableau 4.1 les résume.

Tableau 4.1 Les attributs et les opérations du Moment-Interval
Synthèse à partir de (Peter Coad, Eric Lefebvre et Jeff De Luca, 1999)

Attributs	Utilités
referenceNumber	Génère un numéro de référence pour chaque objet de la classe *Moment-Interval* à un moment ou dans un intervalle de temps.
dateOrDateTimeOrInterval	Responsable de l'enregistrement de la date, du temps ou de l'intervalle de temps d'un objet de la classe *Moment-Interval.*

purpose	Fournit des renseignements sur le but de la classe *Moment-Interval*.
Priority	Fournit l'ordre de priorité des objets dans une même catégorie de classe *Moment-Interval*.
Status	Représente les différents états d'un *Moment-Interval* durant son cycle de vie.
Total	Sert à effectuer des calculs de différents totaux du *Moment-Interval*.
Opérations	**Utilités**
addRemoveDetail	Opération d'ajout et de suppression de détails liés au *Moment-Interval*.
calculateTotal	Opération effectuant des calculs de totaux.
complete	Opération qui sert à marquer une fin réussie de *Moment-Interval.*
cancel	Opération pour marquer un échec ou un abandon de *Moment-Interval*.
generateNextMI	Opération qui crée, initialise et associe des objets qui se produisent suite à l'exécution du *Moment-Interval*.
assessWRTPiorMI	Au cas où le *Moment-Interval* représente une activité planifiée du *Moment-Interval* précédent, cette opération sert à vérifier sa performance par rapport à la planification.
assessWRTNextMI	C'est l'opération inverse de *assessWRTPriorMI*. Elle doit vérifier la performance du *Moment-Interval* à l'égard de la classe qui la suit.
listInstances	Opération capable de rechercher et lister les instances de la classe *Moment-Interval* qui répondent à certains critères.
assessAcrossInstances	Opération permettant d'évaluer un sous ensemble d'instances de *Moment-Interval* selon certains critères.

4.4.2 L'archétype Role

Le deuxième archétype en importance est le *Role*. Il représente la manière de participer d'une partie (personne, organisation), d'une place (endroit) et d'une chose. Il facilite la gestion et

permet la diminution de la surcharge des classes *Party*, *Place* et *Thing* quand ces dernières jouent simultanément différents rôles. Son utilité réside dans l'encapsulation des données et des comportements joués par un rôle. Il est doté d'attributs, de méthodes et d'associations typiques représentant ses responsabilités dans la conception du diagramme de classe d'analyse (Figure 4.5). Un archétype *Role* peut être un caissier, un vérificateur, un chauffeur, etc.

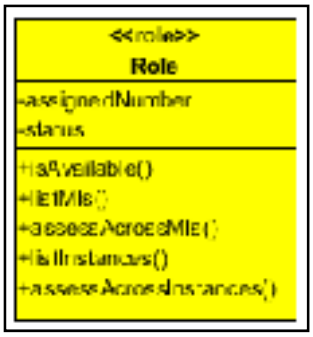


Figure 4.5 L'archétype Role
Empruntée de (Peter Coad, Eric Lefebvre et Jeff De Luca, 1999)

4.4.3 L'archétype Party-Place-Thing

Les archétypes *Party-Place-Thing* (PPT) représentent les principaux acteurs jouant différents rôles. Par exemple, une personne peut être à la fois employé et client. En fait, ce sont des individus, des organisations de toutes sortes (entreprises, organismes, écoles, associations, etc.), des bâtiments ou des sites ainsi que d'autres objets identifiables individuellement qui jouent différents rôles et participent à la consolidation de l'un ou de plusieurs archétypes *Moment-Interval*.

Comme les autres archétypes, l'utilité principale de *Party-Place-Thing* est le renforcement du modèle de classe et la séparation des responsabilités. Celles-ci sont exprimées par des attributs, des opérations et des associations typiques (Figure 4.6).

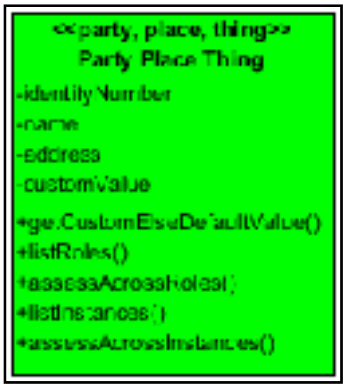


Figure 4.6 L'archétype Party-Place-Thing
Empruntée de (Peter Coad, Eric Lefebvre et Jeff De Luca, 1999)

4.4.4 L'archétype Description

L'archétype *Description* est considéré comme une description d'un catalogue. Il fournit une collection de valeurs et de comportements à toutes les collections d'objets qui correspondent à sa description. Il catégorise et décrit tous les autres types d'archétypes comme la marque et le type d'une voiture ou les dimensions d'une maison. En somme, chaque objet de l'archétype *Description* décrit un ensemble d'objets, de mêmes catégories, des autres archétypes (Figure 4.7).

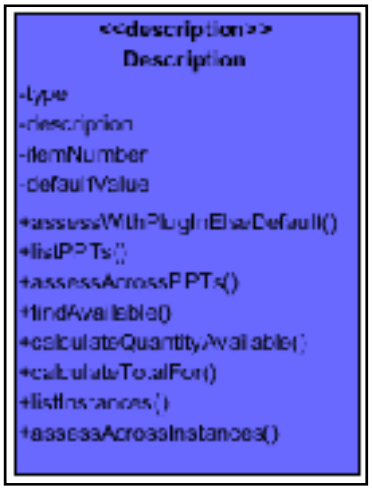


Figure 4.7 L'archétype Description
Empruntée de (Peter Coad, Eric Lefebvre et Jeff De Luca, 1999)

4.5 Domain Neutral Component (DNC)

Un modèle de classe d'analyse peut être construit à l'aide des quatre archétypes. Dans certains cas, autour d'un seul et unique *Moment-Interval* et dans d'autres cas, autour de plusieurs. Selon la règle de la correspondance de l'archétype *Moment-Interval* avec une des activités du modèle de processus d'affaires, les *Moments-Intervals* se suivent, les uns auprès des autres selon l'ordre des enchaînements des activités du modèle de processus d'affaires. Ceci indique qu'il est possible de lier des portions du diagramme de classe d'analyse via les *Moment-Intervals*. Par exemple, un *Moment-Interval* modélisant une livraison serait naturellement lié à un autre *Moment-Interval* modélisant une vente.

Si nous retenons l'idée de l'enchaînement des *Moment-Interval* selon un certain ordre en associant *Moment-Interval* à *PriorMI* et *NextMI* et que nous divisons l'archétype PPT en trois archétypes distincts *Party*, *Place* et *Thing*, en plus d'identifier les archétypes *Role* et *Description*, nous obtiendrons en connectant les 4 archétypes entre eux, un modèle générique appelé (Peter Coad, Eric Lefebvre et Jeff De Luca, 1999) *Domain Neutral Component* (DNC) (Figure 4.8).

Le DNC s'utilise comme un modèle de classe générique qui aide à accélérer le développement des applications logicielles. En fait, il suffit d'identifier la classe *Moment-Interval* et les autres classes archétypes qui y sont reliées. Ensuite, il nous faut remplacer les classes constituant le DNC par les classes identifiées. Quand nous n'avons pas besoin d'une classe du DNC, il suffit de la supprimer pour simplifier le modèle de classe.

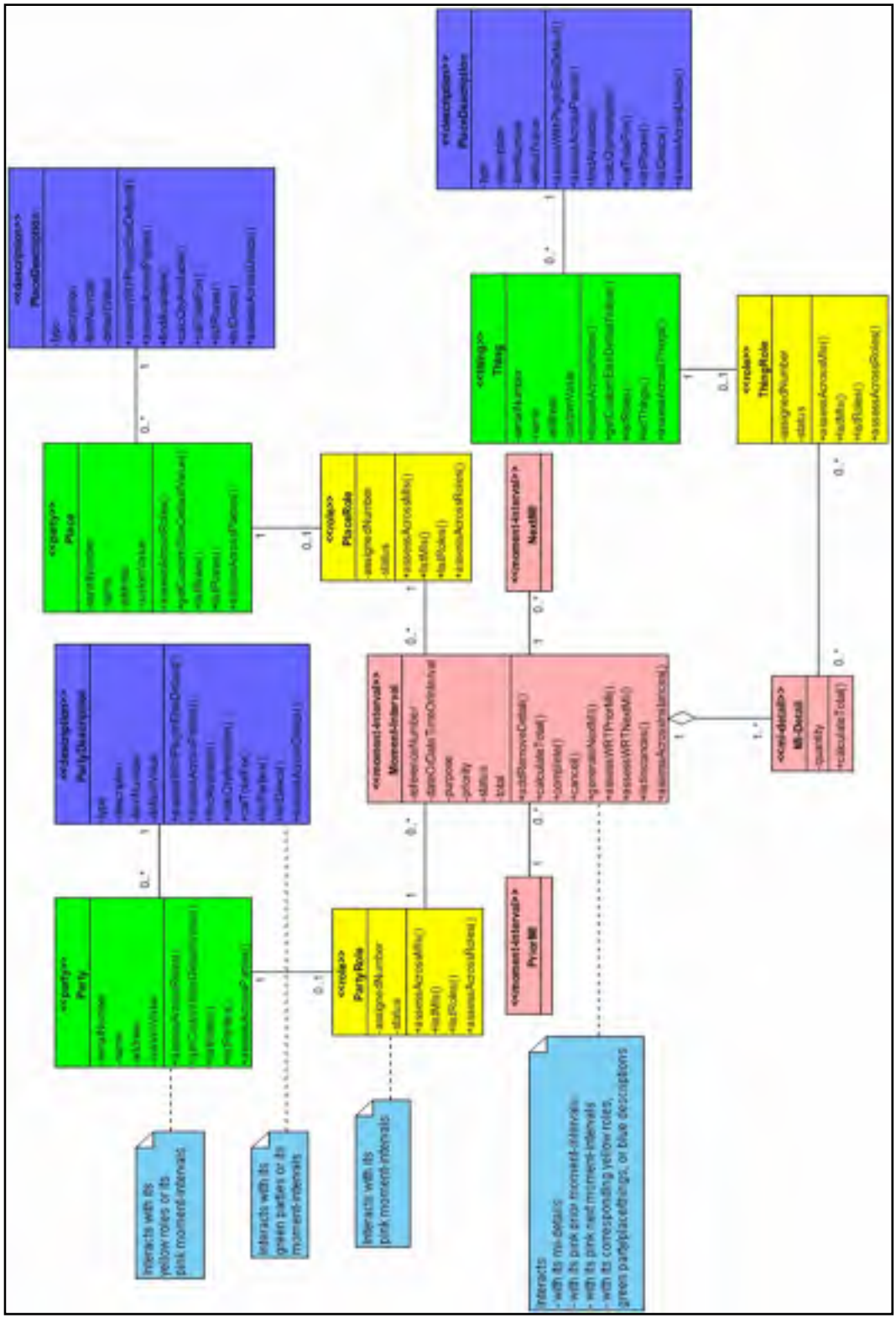


Figure 4.8 Proposition du modèle PIM à l'aide du 'Domaine Neutral Component' (DNC)

4.6 Comment construire le PIM à l'aide des archétypes en se basant sur l'EBP?

Le modèle des processus d'affaires décrit à la fois les activités (l'aspect dynamique) et les objets requis pour la réalisation de ces activités (l'aspect statique). Les objets peuvent être :

- les acteurs responsables de la réalisation de l'activité;
- les ressources requises (humaines, matérielles, informationnelles);
- les produits ou services résultant de l'activité.

Les objets de type acteurs et ressources requises participent à la réalisation de l'activité via un rôle spécifique. La Figure 4.9 illustre le modèle générique du processus d'affaires élémentaires.

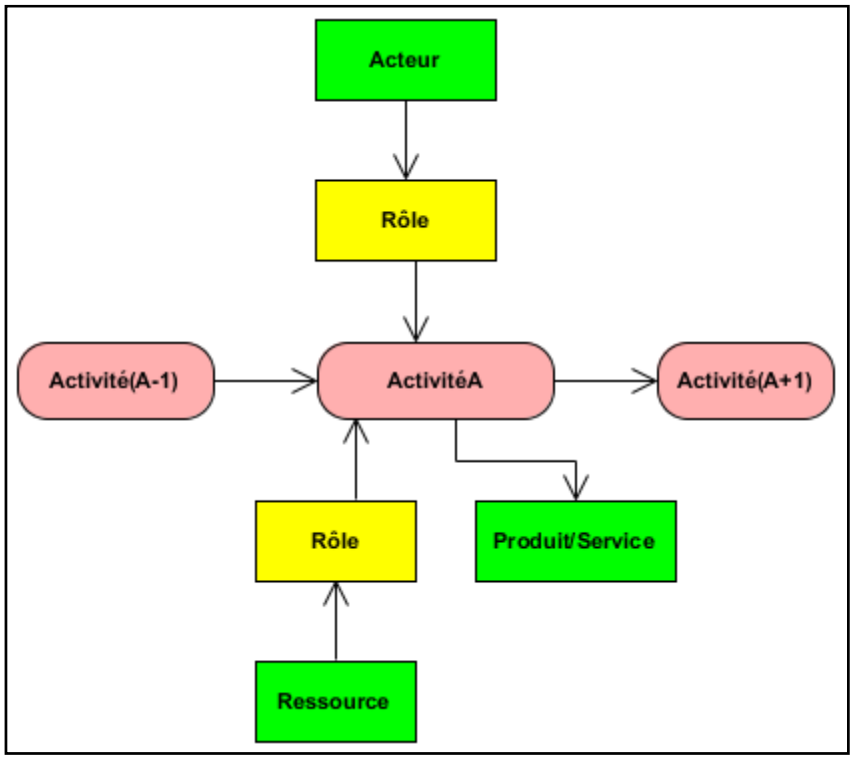


Figure 4.9 Modèle générique d'un EBP

Une fois le modèle des processus d'affaires réalisé, nous définissons les exigences du système pour automatiser les processus concernés (en identifiant les processus automatisables). Pour cela, nous introduisons le système en tant qu'acteur dans le modèle des processus d'affaires. Le système représente l'ensemble des activités du processus d'affaires

que nous désirons automatiser. Cette phase intermédiaire, de passage par un modèle des exigences, est indispensable pour exprimer les choix d'automatisation. En général, ce modèle est représenté sous forme de cas d'utilisation. Ce dernier se définit comme « une description des séquences d'actions qu'un système accomplit et qui donne un résultat observable de valeur pour un acteur particulier »(Kruchten Philippe, 2000). Cette définition se rapproche de celle d'un EBP. D'ailleurs, selon Larman (Larman, 2004), un cas d'utilisation devrait correspondre aux interactions acteur – système d'un EBP.

Donc, un cas d'utilisation spécifie les détails d'un EBP et l'ensemble des cas d'utilisation spécifie le système, c'est-à-dire les activités automatisées du processus d'affaires englobant l'ensemble des EBP correspondants.

Le modèle des cas d'utilisation, qui représente le « quoi » du nouveau système, est transformé en un modèle des composants d'affaires, qui identifie le « comment » structurer le système. Ce modèle, qui sera détaillé dans le chapitre 6, représente une phase intermédiaire entre les modèles des processus d'affaires et des cas d'utilisation et le modèle de classe d'analyse.

Un cas d'utilisation spécifie les actions automatisées d'un EBP. Il comprend en général une activité principale, des ressources requises pour sa réalisation et un produit ou service résultat de l'activité.

Dans l'identification des composants d'affaires, nous en distinguons deux catégories :

1. Les composants d'affaires processus qui correspondent aux activités du processus.
2. Les composants d'affaires entités, qui correspondent aux ressources, acteurs responsables de la réalisation de l'activité et aux produits ou services.

Ainsi, chaque EBP sera transformé en un cas d'utilisation, puis en un composant d'affaires processus qui sera relié à plusieurs composants d'affaires entités. Un composant d'affaires

processus est spécifique à un cas d'utilisation. Un composant d'affaires entités peut participer à plusieurs cas d'utilisation, mais selon un rôle spécifique à chaque cas d'utilisation.

Comme les activités s'enchaînent les unes aux autres selon un ordre particulier, les composants d'affaires processus sont reliés entre eux. Les composants entités sont reliés, selon leur rôle, à chaque composant d'affaires processus qui le requiert.

Le modèle de classe d'analyse (PIM) est représenté par des classes UML. Les composants d'affaires étant identifiés, ils pourront être modélisés par des classes UML en utilisant les quatre archétypes :

- Un composant d'affaires processus = Un archétype *Moment-Interval* ;
- Un composant d'affaires entités = Un ou plusieurs archétypes *PPT* et *Description* ;

L'assemblage des classes d'analyse se fait en suivant le modèle des processus d'affaires tout en se basant sur le DNC. La Figure 4.10 illustre le modèle générique du PIM construit à partir d'un seul EBP. Ce modèle peut être configuré et enrichi par d'autres archétypes selon le nombre des EBP.

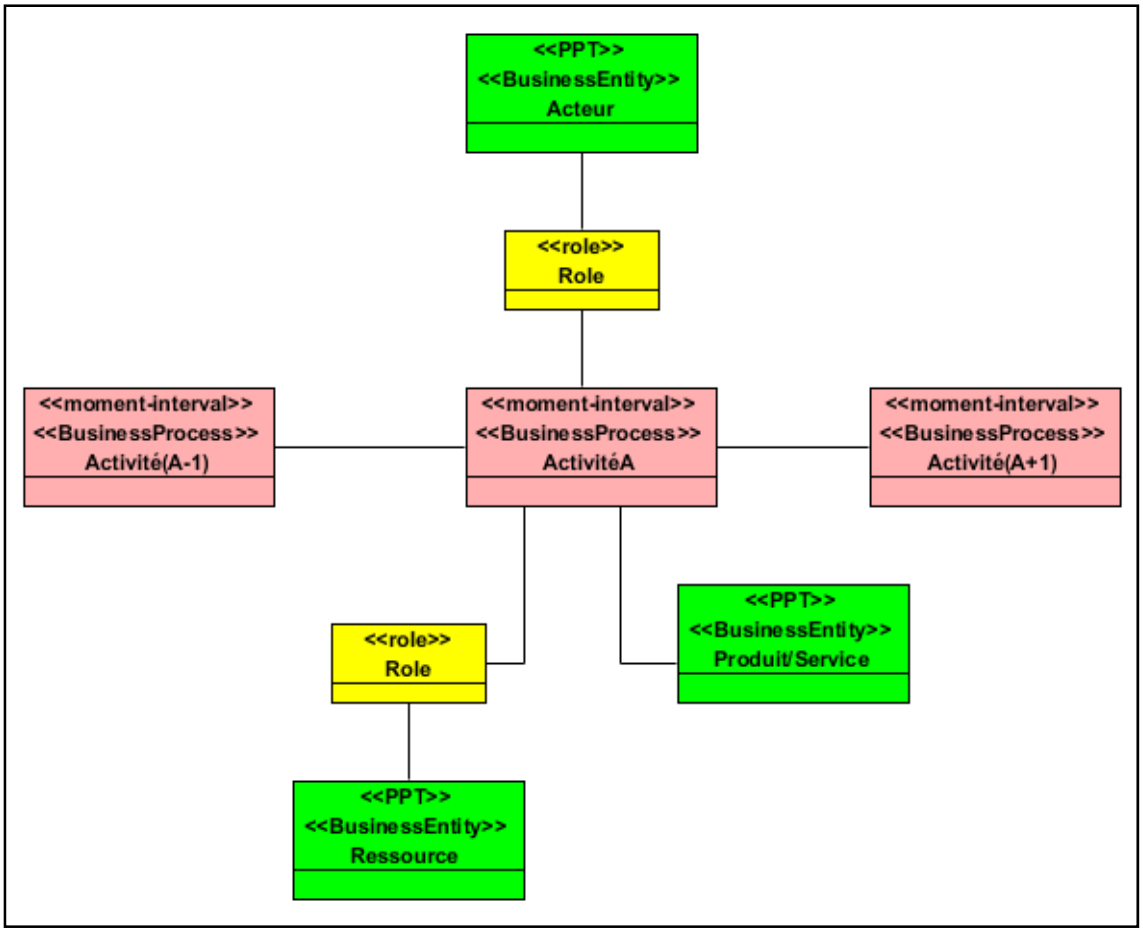


Figure 4.10 Modèle générique du PIM construit sur la base d'un seul EBP

4.7 Méthode de construction du PIM

Pour construire le PIM, nous proposons la méthode décrite ci-dessous, basée sur les EBP, les archétypes et le DNC. Cette méthode utilise la correspondance (illustrée par le signe ↔**)** entre les éléments suivants :

- Un EBP ↔ un composant d'affaires (un seul composant processus + des composants entités) ↔ un seul cas d'utilisation ↔ un diagramme de classe (à l'aide des archétypes).

En voici les étapes :

Suite à l'activité de la modélisation des processus d'affaires :

1) Identifier les EBP dans chaque modèle de processus d'affaires;
2) Identifier dans chaque EBP l'élément dynamique (activité) et les éléments statiques (les objets ressources, produits et services, l'acteur responsable de la réalisation de l'activité).

Suite à la phase des exigences :

3) Décrire chaque EBP candidat à être automatisé par un cas d'utilisation;
4) Compléter les exigences (détailler les cas d'utilisation).

Suite à la phase des exigences nous pouvons construire le modèle de classe d'analyse (PIM) en suivant les étapes suivantes :

5) Définir la(les) classe(s) archétype MI.

Habituellement, une classe MI est associée à l'activité ou l'événement principal d'un EBP. S'il existe plus d'une classe MI, les relier selon leur ordre chronologique. Ensuite, identifier les MI précédentes et les MI suivantes.

6) Sélectionner l'entité d'affaires *MI-Detail* selon les réponses aux questions suivantes :

    Pour chaque classe MI :

        Est-elle reliée au *MI-Detail* ?

        Y a-t-il une entité d'affaires *Party* (une ou plusieurs) ?

        Y a-t-il une entité d'affaires *Place* (généralement une) ?

        Y a-t-il une entité d'affaires *Thing* (une ou plusieurs) ?

    Pour chaque entité d'affaires *Party*, *Place* et *Thing* (PPT) :

        Est-elle reliée à la classe MI ou à la classe *MI-detail* ?

7) Personnaliser les entités d'affaires PPT selon les réponses aux questions ci-dessous :

    Pour chaque entité d'affaires PPT :

        Y a-t-il un archétype *Role* ?

Y a-t-il un archétype *Description* ?

8) Personnaliser les archétypes :

Pour chaque classe :

Sélectionner les attributs et les méthodes génériques pour chaque archétype;

Personnaliser le nom de chaque archétype;

Ajouter des attributs et des méthodes spécifiques selon les besoins.

9) Finaliser le modèle de classe :

Personnaliser le diagramme de classe.

Cette méthode pourrait être supportée par un outil logiciel qui automatiserait ces neuf étapes.

4.8 Étude de cas : réservation d'une chambre d'hôtel

Pour démontrer l'utilité des archétypes et des patrons d'affaires, nous allons appliquer notre méthode de construction du PIM à un simple exemple portant sur la gestion des réservations d'une chambre d'hôtel.

1) L'activité commence par l'identification des processus d'affaires, ce qui nous amène à identifier les EBPs suivants :

- Réservation
- Location (enregistrement à l'arrivée, Check-in)
- Utilisation
- Paiement (Check-out)

Dans ce qui suit, nous allons restreindre notre exemple à l'EBP Réservation d'une chambre.

2) Spécifier les exigences logicielles

Dans cette phase, nous détaillons l'EBP Réservation par un cas d'utilisation. Nous supposons que le client interagit directement comme acteur avec le système. Autrement dit, le commis de l'hôtel jouerait le rôle d'un acteur secondaire, l'intermédiaire entre le client et le système. Voici un extrait du cas d'utilisation Réserver une chambre :

Le scénario nominal est :
Step 1: *The customer calls for a reservation.*
Step 2: *The system opens the reservation function.*
Step 2: The *customer specifies the start and end dates, the category of desired rooms.*
Step 3: *The system checks the availability for the required period and the room category.*
Step 4: *The customer accepts.*
Step 5: *The system asks for a credit card number.*
Step 6: *The customer indicates his (her) credit card number.*
Step 7: *The system confirms with a reservation number.*

Le scénario alternatif peut se déclencher quand la chambre demandée par le client est indisponible.

3) Construire le diagramme de classe de l'EBP Réservation.

La Réservation représente l'activité (Activity) de l'EBP et est donc définie par l'archétype *Moment-Interval*.

Si le client demande la réservation de plusieurs chambres, les détails de la Réservation sont définis par *MI-Detail* (un détail pour chaque chambre réservée). Comme la Location représente l'EBP qui suit l'EBP Réservation, elle sera représentée par l'archétype *Next-MI*.

Les entités d'affaires sont Client, Employé et Chambre. Ils sont définis par les archétypes *Party-Place-Thing* comme suit :

- *Party* : Client, Employé (L'archétype *Role*).
- *Place* : Chambre (l'archétype *Thing* et l'archétype *Description* décrivant la chambre).

L'analyse du processus d'affaires et des entités d'affaires qui y sont reliées nous emmène à construire le diagramme de classe d'analyse (PIM) comme illustré à la Figure 4.11.

La classe *Description* a été complétée par deux classes, la classe Prix et la classe Taxe qui sont également représentées par l'archétype *Description*. La classe Carte de Crédit est liée à la classe Client et représentée par l'archétype *Thing*.

Ce modèle PIM de la réservation d'une chambre d'hôtel peut être complété par les attributs et les méthodes génériques offerts par chacun des archétypes.

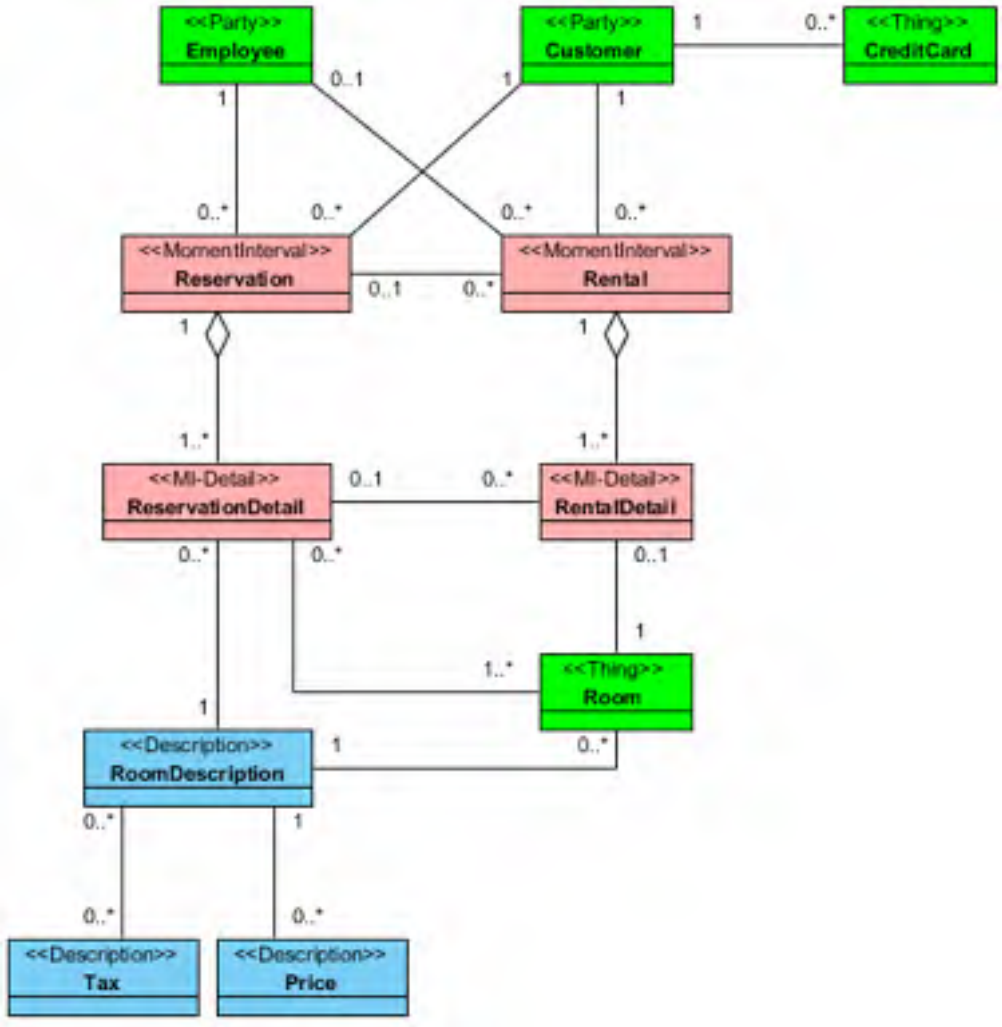


Figure 4.11 Diagramme de classe (PIM) de la réservation d'une chambre d'hôtel

4.9 Discussion

Ce chapitre présente une proposition de l'architecture du PIM ainsi qu'une méthode permettant sa construction.

L'utilisation des archétypes et du DNC permet de guider, étape par étape, la création du modèle PIM. Cette méthode offre la possibilité de développer rapidement des applications logicielles en utilisant des catalogues réutilisables de composants d'affaires.

CHAPITRE 5

CONCEPTION DU MODÈLE DE PROCESSUS D'AFFAIRES POUR LE CIM

5.1 Conception du Business Process Model

Ce chapitre présente une partie de la phase 4 de ce projet de recherche, soit la conception du modèle des processus d'affaires (BPM).

5.1.1 Anatomie du BPM

En général, nous construisons le modèle des processus d'affaires en décomposant ceux-ci jusqu'aux processus d'affaires élémentaires (Elementary Business Process, ou EBP), c'est-à-dire jusqu'au niveau où nous ne sommes plus intéressés à décomposer davantage.

Un EBP se définit comme : "*A task performed by one person in one place at one time, in response to a business event, which adds measurable business value and leaves the data in a consistent state*" (Larman, 2004).

Un modèle des processus d'affaires se compose donc d'un ensemble d'EBP qui s'enchaînent selon un *workflow* spécifiant la transmission du travail d'un acteur à l'autre.

Le modèle des processus d'affaires décrit à la fois les activités (l'aspect dynamique) et les objets requis pour la réalisation de l'activité (l'aspect statique). Les objets peuvent être :

- Les acteurs, responsables de la réalisation;
- Les ressources requises (humaines, matérielles, informationnelles);
- Les produits ou services résultant de l'activité.

En résumé,

- Un EBP a un acteur responsable de sa réalisation;
- Un EBP consomme des ressources;
- Un EBP produit un résultat qui peut être soit un service, soit un produit.

5.1.2 Le patron EBP

Pour une utilisation plus générique du concept de l'EBP, de ses relations avec les objets et l'acteur responsable de son exécution, il est primordial de définir un patron nommé patron EBP. En effet, en définissant un patron EBP, nous pouvons incorporer les quatre éléments - l'acteur, les ressources et les résultats de type produit ou service ainsi que les relations entre eux - dans un seul concept.

Dans l'élaboration du patron EBP, nous avons décidé, pour des raisons de simplification et pour ne pas confondre l'EBP et le patron EBP, de renommer l'EBP par le mot ***Activity***.

L'*Activity* fait référence à l'élément *Activity* du diagramme d'activité qui est un comportement en soi et peut être associé à des paramètres. Une *Activity* définit un comportement décrit par une séquence organisée d'unités dont les éléments simples sont les actions.

La Figure 5.1 montre deux représentations possibles du patron EBP.

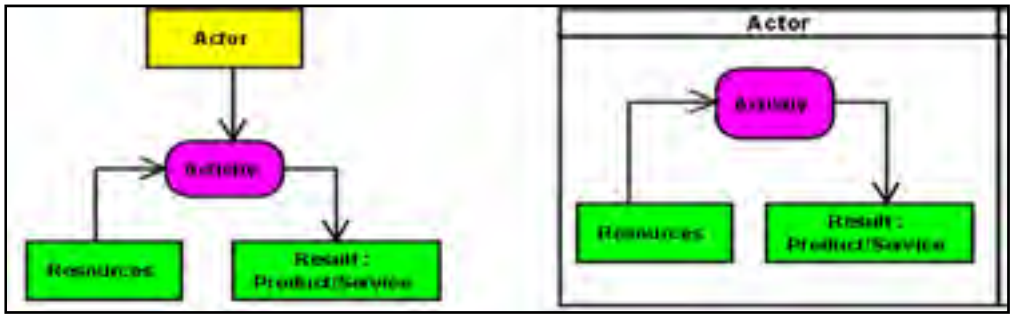


Figure 5.1 EBP Pattern : les deux représentations possibles

Nous pouvons donc représenter un métamodèle (Meta-Pattern) du patron EBP comme suit (Figure 5.2) :

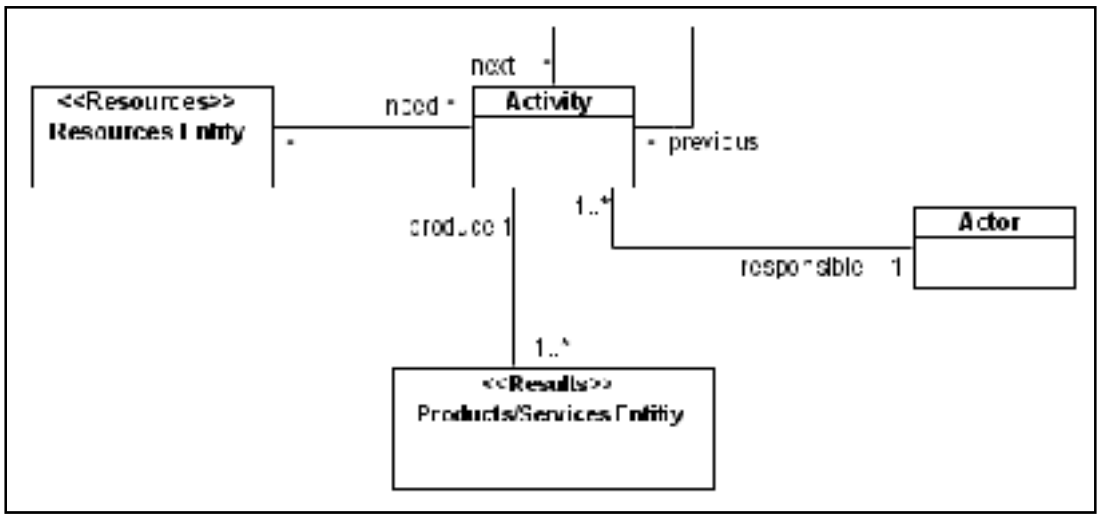


Figure 5.2 EBP pattern Metamodel (Meta-Pattern)

5.1.3 Règles de cohérence du patron EBP

Pour donner un cadre plus exhaustif au patron EBP, il est important de s'assurer qu'il remplisse les contraintes suivantes :

RC-EBP1 : un Acteur est responsable d'une ou plusieurs *Activities* ;
RC-EBP2 : une *Activity* utilise une *Resource* ;
RC-EBP3 : une *Resource* peut être utilisée par une ou plusieurs *Activities* ;
RC-EBP4 : une *Activity* produit un Produit ou un Service.

5.1.4 Catégorisation des patrons EBP

Dans (Dijkman et Joosten, 2002; Johann Eder et Walter Liebhart, 1996), les auteurs distinguent trois types de tâches-processus : une tâche automatique, une tâche supportée et une tâche manuelle. Dans notre cas, la tâche-processus représente l'*Activity* du patron EBP.

Cette identification servira à identifier à la fois les cas d'utilisation à automatiser et les composants d'affaires à concevoir. Dans le même ordre d'idées que (Dijkman et Joosten, 2002), nous adoptons cette catégorisation des tâches-processus et nous l'appliquons aux *Activities* des patrons EBP. L'élément *Activity* représente l'élément dynamique du patron EBP qui joue un rôle décideur dans sa classification :

- **L'*Activity* automatique** est une *Activity* qui peut être, une fois implémentée, exécutée par un système informatique sans l'intervention ou l'interaction avec une entité externe (acteur). Un exemple d'*Activity* automatique est la génération automatique d'une lettre.

- **L'*Activity* supportée** est une *Activity* qui peut être, une fois implémentée, exécutée par le système informatique à l'aide ou en interaction avec une entité externe (acteur). Un exemple d'un EBP supporté est la réservation d'une voiture, qui demande l'obtention des informations sur la disponibilité des voitures dans le système informatique de l'agence de location (la base de données de l'agence de location de voitures).

- **L'*Activity* manuelle** est une *Activity* qui peut être réalisée complètement par une entité externe du système (acteur). Elle est complètement sous le contrôle d'un agent humain. Par conséquent, cette *Activity* ne peut pas être automatisée et ne peut pas être représentée dans un système informatique. Toutefois, elle doit être prise en compte lors des spécifications des exigences logicielles. Elle pourra être représentée, à titre d'exemple, par une règle d'affaires. Un exemple d'un EBP manuel est l'écriture d'une lettre, l'envoi de fax ou la réalisation d'une photocopie.

5.1.5 Sélection des *Activities*

Dans notre recherche, il faut préciser que nous nous intéressons uniquement aux *Activities* supportées, car notre objectif vise à en extraire des cas d'utilisations. Seuls les *Activities* supportées nous offrent la possibilité d'avoir une interaction entre un acteur et le système, ce qui justifie la définition d'un cas d'utilisation. Cette limitation aux *Activities* supportées nous

facilite le processus d'identification des cas d'utilisation à partir du modèle des processus d'affaires.

En revanche, il est aussi primordial de prendre en compte les *Activities* automatiques, car d'une façon ou d'une autre, elles seront prises en compte lors du développement du logiciel.

Quant aux *Activities* manuelles, elles ne seront pas prises en compte dans le BPM et ne seront pas abordées dans notre recherche.

Quand une *Activity* est manuelle, elle ne sera pas non plus représentée ni par un composant d'affaires, ni par un cas d'utilisation. Par définition, elle ne remplit pas la condition d'interaction acteur-système d'un cas d'utilisation et il n'est donc pas intéressant de la représenter par un composant d'affaires sauf à des fins de documentation. L'architecte d'affaires peut décider d'incorporer ce cas d'utilisation manuel dans le cas d'utilisation qui le suit ou qui le précède sous forme d'une condition ou d'une règle d'affaires. En général, les *Activities* manuelles ne sont pas supportées par le système informatique.

Quand une *Activity* est automatique, elle sera décrite par un cas d'utilisation spécifique dont l'acteur sera représenté par un ''Déclencheur Événementiel''. Ce dernier peut être un bouton poussoir, un temporisateur, etc. Il sera représenté par un composant d'affaires, car il fera partie intégrante du système informatique à développer. Il est aussi préférable de laisser le choix à l'architecte de décider de son sort, à savoir l'incorporer dans le cas d'utilisation qui le précède ou dans celui qui le suit. Dans certains cas, l'*Activity* automatique peut être simplement représentée par une règle d'affaires.

5.1.6 Conception du BPM et de son métamodèle

Notre choix s'est porté sur une représentation généralement reconnue du modèle des processus d'affaires, qui est celle du diagramme d'activités telle que spécifiée par UML2.

UML offre la possibilité d'étendre et d'adapter son métamodèle à un domaine particulier par la création de profils. Un profil UML peut étendre un métamodèle ou un autre profil tout en préservant la syntaxe et la sémantique des éléments existants d'UML. Il ajoute des éléments qui étendent les classes existantes. Un profil UML fournit des stéréotypes, des contraintes et des *Tagged values*. Il caractérise le domaine dans lequel il est défini.

Pour adapter le diagramme d'activité UML2 à notre démarche de conception du BPM, un profil UML de BPM est proposé à la Figure 5.3. Il est basé sur le métamodèle du diagramme d'activité UML2 et prend en compte les éléments du patron EBP.

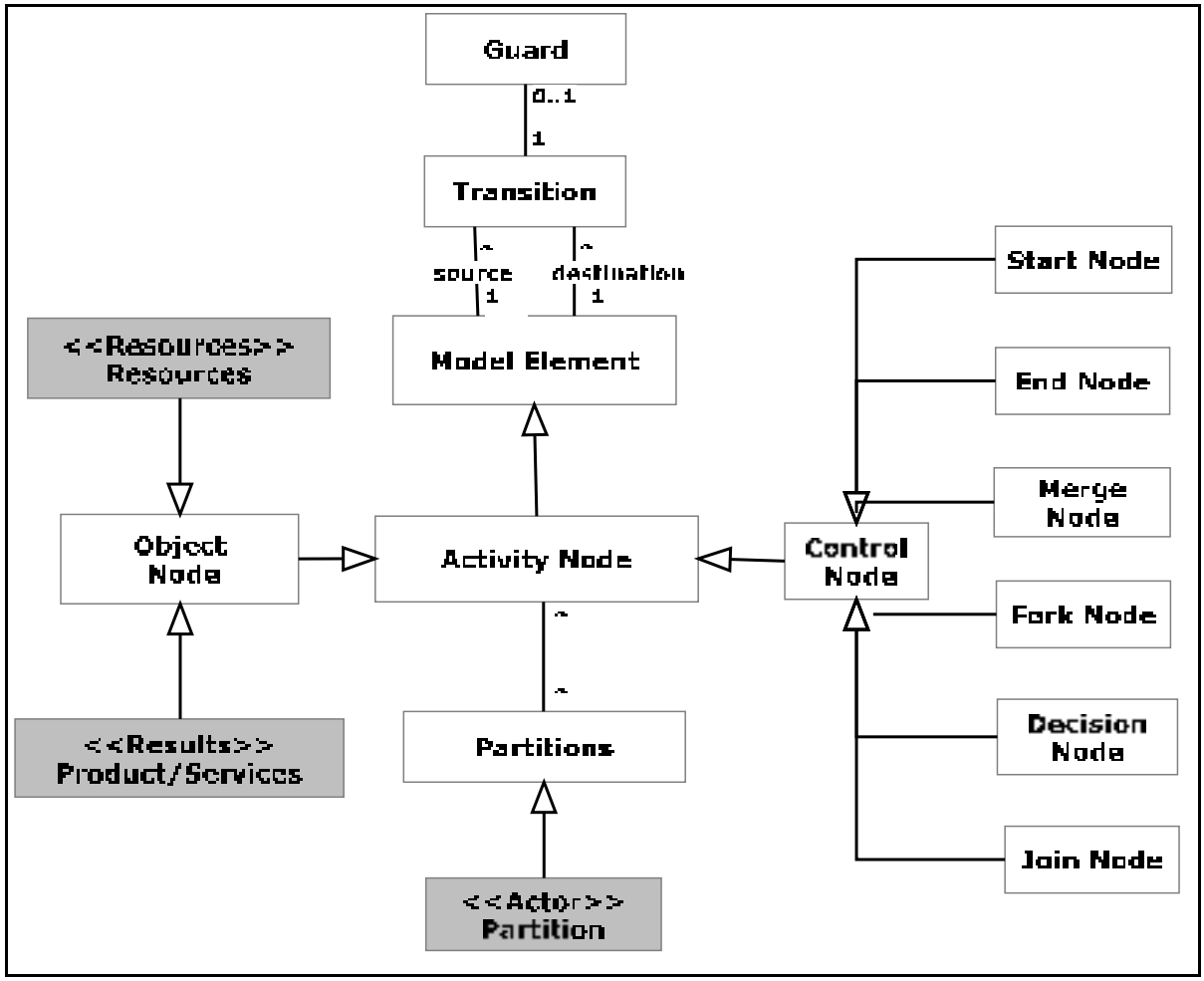


Figure 5.3 Profil BPM basé sur le métamodèle UML2 du diagramme d'activité

Dans le profil UML du BPM, nous avons créé deux nouveaux éléments de type *Object Node* spécifiés par deux stéréotypes différents, *Resources* et *Results* :

- Le stéréotype ‘’Resources*’’* désigne l’objet qui fait référence à des ressources du patron EBP,
- Le stéréotype ‘’Results’’ désigne l’objet qui fait référence aux produits/services du patron EBP.

Ces deux nouveaux objets stéréotypés étendent l’élément *Object Node* du métamodèle du diagramme d’activité UML2.

L’OMG a défini l’*Object Node* comme une ‘’ *activity node that indicates an instance of a particular classifier, possibly in a particular state, may be available at a particular point in the activity. Object nodes can be used in a variety of ways, depending on where objects are flowing from and to*’’ (OMG, 2009-02-02). Par conséquent, les *Object Nodes* représentent des instances concrètes des informations d’objets, qui sont des paramètres d’entrée ou de sortie de l’*Activity* du patron EBP.

L’élément *Activity* du patron EBP est représenté par l’élément *Activity Node* dans le profil UML2 du diagramme d’activité. L’élément *Activity Node* spécifie la coordination de l’exécution des comportements subordonnés.

Nous avons également étendu l’élément Partitions du métamodèle du diagramme d’activité UML2 par l’élément de type Partition stéréotypé *Actor*. Ceci sert à spécifier l’acteur responsable de chaque EBP du patron EBP dans le BPM.

Le Tableau 5.1 dresse les trois stéréotypes que nous avons ajoutés au profil UML du modèle des processus d’affaires.

Tableau 5.1 Stéréotypes du profil UML ajoutés pour les processus d'affaires

Stéréotype	Appliqué à	Définition
<<Resources>>	Object Node	Indique que l'objet représente une ressource utilisée par l'*Activity* du patron EBP
<<Results>>	Object Node	Indique que l'objet représente un produit ou un service produit par l'*Activity* du patron EBP
<<Actor>>	Partition	Indique que l'objet représente l'acteur responsable de l'*Activity* du patron EBP

5.1.7 Éléments du modèle des processus d'affaires et contraintes de conception

Le modèle des processus d'affaires est constitué des mêmes éléments illustrés par la Figure 5.3, soit le profil BPM basé sur le métamodèle du diagramme d'activités. La sémantique et la syntaxe du diagramme d'activités UML2 servent de base à la conception du BPM.

Pour alléger le diagramme et rendre la modélisation du BPM plus intuitive, nous avons décidé d'apporter quelques modifications mineures à quelques éléments du BPM :

- L'élément *Object Node* stéréotypé *Resources* sera réduit en un élément de type *Object Node* sans aucun stéréotype. Un arc de couleur bleu sortant de l'*Object Node* et entrant dans l'*Activity Node* remplacera le stéréotype *Resources*.

Figure 5.4 Remplacement du stéréotype *Resources* dans l'*Object Node*

- l'élément *Object Node* stéréotypé «Results» sera réduit en un élément de type *Object Node* sans aucun stéréotype. Un arc de couleur rouge sortant de l'*Activity Node* et entrant dans l'*Object Node* remplacera le stéréotype « Resources ».

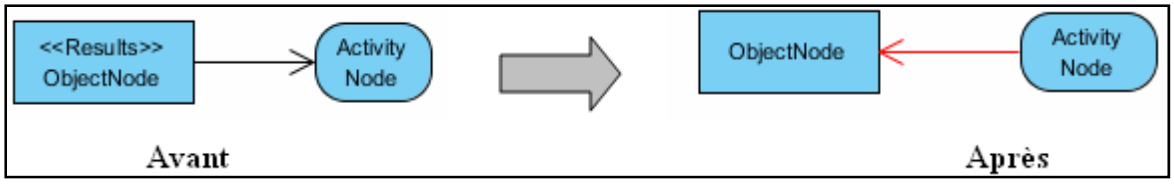


Figure 5.5 Remplacement du stéréotype *Results* dans l'*Object Node*

- L'élément partition «Swimlane» ne sera pas stéréotypé «Actor». L'élément partition signifie d'une façon directe qu'il s'agit d'un acteur.

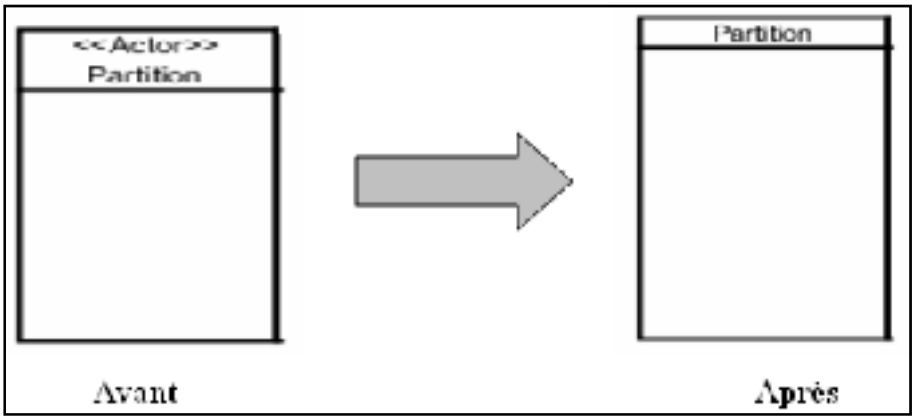


Figure 5.6 Remplacement du stéréotype *Actor* dans l'élément Partition

5.1.8 Procédure de conception du BPM

La procédure que nous proposons pour construire le BPM inclue les cinq étapes suivantes :

P1 : Identifier les EBP, les objets ressources, les objets résultats et les acteurs;
P2 : Identifier la responsabilité de chaque acteur. Un acteur doit être responsable de un ou plusieurs EBP;
P3 : Relier les EBP entre eux selon leur enchaînement;
P4 : Relier les objets ressources et les objets résultats aux EBP selon la règle suivante : un EBP consomme ou plusieurs objets ressources et produit un ou plusieurs objets résultats;
P5 : Compléter le BPM en utilisant le profil BPM proposé.

5.2 Conclusion

Ce chapitre présente une nouvelle technique de conception du modèle de processus d'affaires. Celle-ci se base essentiellement sur l'identification et la catégorisation des processus d'affaires élémentaires (EBP). Le BPM, sera ensuite conçu selon l'enchaînement des EBP. Un profil du BPM, basé sur le métamodèle du diagramme d'activité UML2, a été également proposé.

CHAPITRE 6

TRANSFORMATION DU <<BUSINESS PROCESS MODEL>> (BPM) EN <<BUSINESS PROCESS COMPONENT>> (BCM)

6.1 Introduction

Ce chapitre présente une partie de la phase 4 de ce projet de recherche, soit la conception du modèle des composants d'affaires (BCM) ainsi que la démarche permettant de transformer le BPM en BCM.

Théoriquement, les applications logicielles devraient soutenir les processus d'affaires. Une des causes fréquentes des échecs de développement de projets logiciels est l'inadéquation entre les exigences d'affaires et les fonctionnalités logicielles livrées. Deux phénomènes opposés peuvent être observés (Boris Shishkov, 2002) :

- D'une part, développer un logiciel sans investiguer les processus d'affaires qu'il devra supporter. Cette situation signifie que les exigences d'affaires sont mal déterminées et que le modèle de conception du logiciel ne s'appuie pas sur le modèle des processus d'affaires. Ainsi, le logiciel développé ne supporte pas adéquatement les processus d'affaires. Bien que, parfois, la qualité de ce logiciel pourrait être acceptée du point de vue technique, le niveau de support du logiciel vis-à-vis des processus d'affaires resterait faible;
- D'autre part, dans de nombreux cas, la modélisation des processus d'affaires est réalisée avant la phase de la conception du logiciel. Le modèle des processus d'affaires n'est utilisé que partiellement, car il n'est pas complètement transformable en modèle de conception et n'en représente pas une base pertinente. Cela ne permet pas d'exploiter la pleine utilité prévue du logiciel.

Par conséquent, la phase de la modélisation des processus d'affaires et la phase de conception logicielle doivent être considérées comme deux activités complémentaires. Dans

le sens où le modèle de conception logicielle doit prendre en compte les processus d'affaires et les supporter intégralement. En d'autres mots, le modèle de conception logicielle doit refléter, représenter et transcrire les processus élémentaires du modèle des processus d'affaires. Une traçabilité bidirectionnelle entre le modèle des processus d'affaires et le modèle de conception logicielle s'impose.

De nombreuses recherches ont abordé les questions liées à ce problème. La plupart de ces recherches sont des représentations formelles pour décrire les processus d'affaires, mais ne sont pas davantage liées à la construction du modèle de conception logicielle. Olivera *et al.,* ont contribué à cette direction en proposant la conception de logiciels sur la base de l'analyse des exigences d'affaires (Toacy Cavalcante de Oliveira et al., Using XML and Frameworks to Develop Information Systems). Leur approche est un pas en avant même si elle ne couvre pas une transformation du modèle des processus d'affaires en un modèle de conception. Hikita *et al.,* (Hikita et Matsumoto, 2001) ont étudié comment l'apparence des exigences additionnelles pourraient se refléter sur le cycle de développement logiciel. C'est aussi un résultat prometteur, mais qui ne permet pas de résoudre complètement le problème. Krutchen (Jacobson, 1992) introduit des cas d'utilisation d'affaires basés sur le concept existant des cas d'utilisation que nous considérons utiles pour coupler le modèle des processus d'affaires et le modèle de conception (Kruchten Philippe, 2000). Mais la question qui se pose encore est de savoir comment identifier un tel cas d'utilisation.

Par conséquent, nous pouvons conclure que nous avons encore besoin de connaissances pour construire systématiquement le modèle de conception logicielle sur la base du modèle des processus d'affaires.

Une approche prometteuse, fondée sur les principes de l'orienté-objet, consiste à développer des applications logicielles à base des composants d'affaires (Jacobson, 1992). Elle est largement utilisée comme une approche particulière pour la construction des modèles des systèmes complexes dans lesquels un système se compose d'un grand nombre d'objets. Cette approche ne s'applique pas seulement à la phase de conception logicielle, mais aussi à la

phase des exigences d'affaires (Ivar Jacobson, Maria Ericsson et Agneta Jacobson, 1995). Ainsi, il semble possible que la construction du modèle de conception logicielle (PIM) et le modèle des processus d'affaires puissent être rapprochés en se basant sur les composants d'affaires qui sont dérivés de certains processus d'affaires élémentaires. Ces composants doivent réduire le fossé entre les deux constructions de modèles mentionnés. Une fois les composants d'affaires identifiés, ils pourraient être réutilisables, remplaçables et interopérables. Ils peuvent aussi permettre la construction des applications qui possèdent une configuration durable et un haut degré de flexibilité et de maintenabilité. Grâce à cette approche, le cycle de développement logiciel serait également amélioré pour concevoir de futures applications en utilisant des composants d'affaires déjà développés.

Pour ces raisons, et en considérant le problème de l'alignement entre le modèle des processus d'affaires et le modèle de conception, nous mettons un accent particulier sur l'identification et la génération des composants d'affaires à partir des processus d'affaires.

6.2 Construction des composants d'affaires

L'utilité principale des composants d'affaires est leur réutilisation dans une problématique donnée en cours de développement, et possiblement dans d'autres domaines par la suite. Le défi de l'architecte d'affaires dans ce cas repose sur l'identification des composants d'affaires qui peuvent être développés de manière efficace, à faible coût, facile à réutiliser, à assembler et à maintenir. L'utilisation des composants d'affaires présente une capacité à personnaliser les applications par leur sélection et leur assemblage.

Identifier des artefacts réutilisables est reconnu comme une des tâches les plus difficiles dans le domaine de la réutilisation des logiciels (Uday Apte et al., 1990). Bien que les problèmes liés à la réutilisation des artefacts logiciels comme le code et les composants de conception logiciels (EJB, .NET) soient bien traités dans la littérature, les problèmes de l'identification, de la fabrication et de la réutilisation des composants d'affaires ne sont pas encore suffisamment pris en compte (Makrygiannis, 1998).

Les composants d'affaires sont différents des artefacts logiciels traditionnels et le processus de leur conception doit donc tenir compte de cette différence. Par exemple, les artefacts logiciels traditionnels (ex : segment de code, objets, etc.) sont généralement de granularité fine et représentent un faible niveau de représentation d'un domaine d'affaires. D'autre part, les composants d'affaires sont d'une granularité élevée et sont destinés à fournir un haut niveau de représentation orienté-affaires d'un domaine d'affaires. Toutefois, les composants d'affaires, grâce à leur granularité élevée, permettent aux architectes d'affaires d'identifier les composants qui répondent à leurs exigences d'affaires, et ensuite de les assembler à large échelle dans des applications logicielles.

6.2.1 Définitions : Qu'est ce qu'un composant d'affaires ?

Un composant d'affaires est un bloc modulaire qui compose une partie des affaires de l'entreprise.

Un composant d'affaires comprend cinq dimensions (George Pohle, Peter Korsten et Shanker Ramamurthy, 2005) :

- son objectif, qui est la raison logique de son existence au sein de l'organisation, se définit par la valeur qu'il offre à d'autres composants d'affaires;
- chaque composant effectue une série d'activités pour atteindre son objectif;
- les composants ont besoin des ressources, des agents, des connaissances et des actifs qui supportent leurs activités;
- chaque composant est géré comme une entité indépendante basée sur son modèle de gouvernance;
- un composant d'affaires fournit et reçoit un service d'affaires.

Un composant d'affaires peut se définir comme un composant qui fournit un certain ensemble de services à un domaine d'application d'affaires. Des domaines d'applications d'affaires typiques sont le domaine bancaire, le domaine des assurances, le domaine de la vente ou de l'industrie manufacturière.

Une autre définition nous semble importante (Boris Shishkov et Jan L. G. Dietz, 2004) : ‘’*A Business Component is a model in which a Business Process is modeled. The model should be characterized by a good representation of the Business Process*’’. D’après cette définition, nous retenons qu’un modèle de composant d’affaires devrait représenter le modèle des processus d’affaires dans un autre niveau de granularité d’affaires. Dans notre recherche, le modèle des processus d’affaires et le modèle des composants d’affaires sont tous les deux une composante du CIM. D’ailleurs, nous proposons la définition suivante du composant d’affaires et du modèle de composant d’affaires : ‘’Un composant d’affaires est un composant qui modélise un processus d’affaires élémentaires. Cette modélisation représente une vue globale de l’ensemble des éléments à développer. Le modèle des composants d’affaires est un modèle intermédiaire qui agit comme un connecteur entre le modèle des processus d’affaires, le modèle de cas d’utilisation, le modèle des composants système et le modèle de classe.’’

6.2.2 Identification des composants d’affaires

L’identification des composants d’affaires est reconnue comme une des phases les plus difficiles dans le processus du développement de logiciel à base de composants. Les méthodes actuelles d’identification des composants sont basées sur le *Clustering analysis* (Hemant Jain et al., 2001; Lee et al., 2001), *Matrix analysis* ’(Ganesan et Sengupta, 2001; Sang Duck et al., 1999) et *Graph-based decomposition methods* (Brian S. Mitchell, 2003; Mancoridis et al., 1999; Yves Chiricota, Fabien Jourdan et Guy Melançon, 2003). Ces méthodes discutent principalement de la conception des composants orientés-objets d’UML et se focalisent sur une représentation orientée technique du domaine. Cependant, les composants d’affaires sont à granularité élevée et sont supposés fournir un haut niveau de représentation d’un domaine orienté-affaires.

Dans cette étape de recherche, nous nous focalisons principalement sur l’identification des composants d’affaires à partir d’un modèle de processus d’affaires. Nous considérons que la pertinence et la qualité du modèle de processus d’affaires sont des facteurs déterminants dans

le processus de l'identification des composants d'affaires, à savoir des composants d'affaires réutilisables, facile à assembler et à maintenir.

6.2.3 Catégorisation des composants d'affaires

Concevoir un modèle de composants d'affaires sur la base d'un modèle de processus d'affaires nécessite l'identification, la catégorisation des composants et une méthode pour les assembler. Dans cette section, nous introduisons la catégorisation des composants d'affaires en composants-processus et en composants-entités selon leur nature dynamique ou statique.

Dans un modèle de composants d'affaires, il y a deux importants éléments qui contribuent à sa conception. Il y a des éléments dynamiques qui, à travers plusieurs calculs et opérations, permettent de résoudre un problème. Il y a également des éléments statiques qui sont des éléments réels faisant partie du problème à résoudre. Les éléments statiques sont utilisés par les éléments dynamiques afin d'apporter une solution au problème. Par exemple, l'opération "Retrait d'argent" est un processus élémentaire ou un cas d'utilisation dans une application bancaire. Cette opération utilisera un guichet automatique, qui est un objet du domaine, comme un outil lui permettant d'accomplir la requête du client. Ces deux types d'éléments englobent à eux seuls le concept de composant que nous pouvons généralement trouver dans un problème à résoudre (Emmanuel Renaux, 2004; Jack Greenfield, 2004).

Nous avons déjà spécifié la catégorisation de l'EBP selon deux aspects : dynamique et statique. L'aspect dynamique représente l'*Activity* du patron EBP et l'aspect statique représente les objets requis pour la réalisation de l'*Activity* du patron EBP. Les objets peuvent être :

- Les acteurs, responsables de la réalisation;
- Les ressources requises (humaines, matérielles, informationnelles);
- Les produits ou services résultants de l'*Activity*.

C’est selon cette catégorisation que nous proposons deux types de composants d’affaires. Chaque type de composant sera basé sur la nature de l’aspect de l’élément qui constitue le patron EBP, soit un aspect dynamique ou un aspect statique. Cette catégorisation permet de garder une traçabilité entre les éléments statiques et dynamiques dans la transformation du BPM en BCM. Cette catégorisation selon l’aspect, permettra de faciliter la maintenance et de mieux réutiliser les EBP et les composants d’affaires.

Dans cette étude, les éléments dynamiques du modèle de composants d’affaires seront appelés les composants-processus ‘’Process Component’’ et les éléments statiques, les composants-entités ‘’Entity Component’’. Les termes utilisés ont déjà été employés dans d’autres études (Ivar Jacobson, 2005; Peter Coad, 1999).

6.2.3.1 Composant-processus (Process Component)

Le composant-processus est le composant central qui expose les opérations, les services et les objets d’affaires utilisés. Un composant-processus est une suite d’opérations initiée par une entité, tel un acteur d’un cas d’utilisation ou d’un EBP du modèle des processus d’affaires. C’est une fonctionnalité existante d’une application ou d’un système. Il est cependant indépendant du contexte de son utilisation.

6.2.3.2 Composant-entité (Entity Component)

Un composant-entité est un élément qui fait partie du problème du domaine d’affaires. C’est une entité statique spécifiquement liée au domaine du problème d’affaires que nous voulons résoudre. Par exemple, un "guichet automatique" est spécifique au domaine d’affaires bancaire et est un composant du domaine dans une application de type bancaire. Les composants-entités ne communiquent généralement pas entre eux directement, car ils sont utilisés et gérés par les composants-processus.

6.2.4 Patron du composant d'affaires

À partir des deux composants d'affaires que nous avons identifiés, nous proposons un patron de composant d'affaires le BC pattern (Figure 6.1) et son métamodèle (Figure 6.2). Le BC pattern est basé sur la même structure du patron EBP.

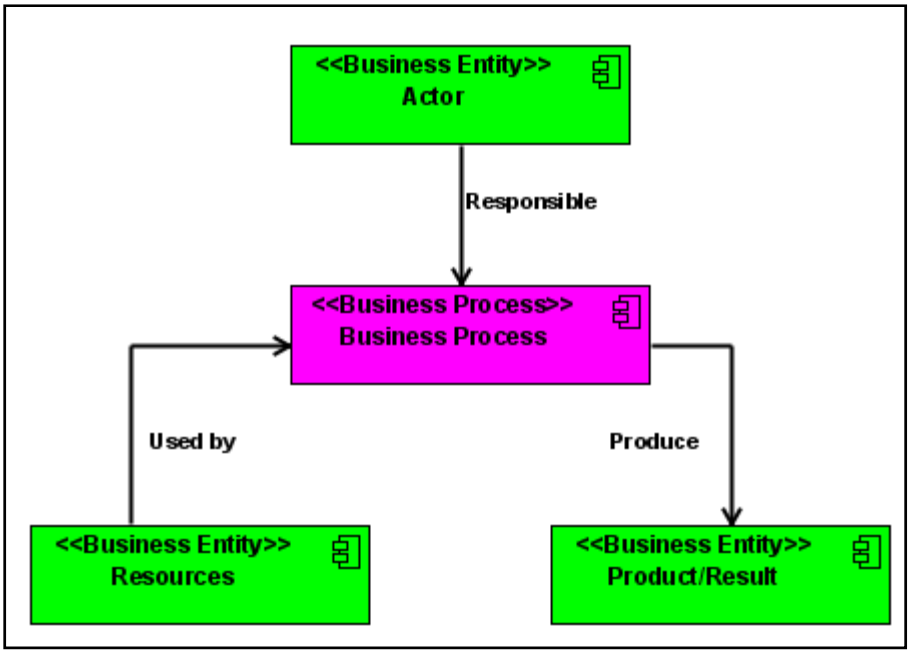


Figure 6.1 Business Component pattern

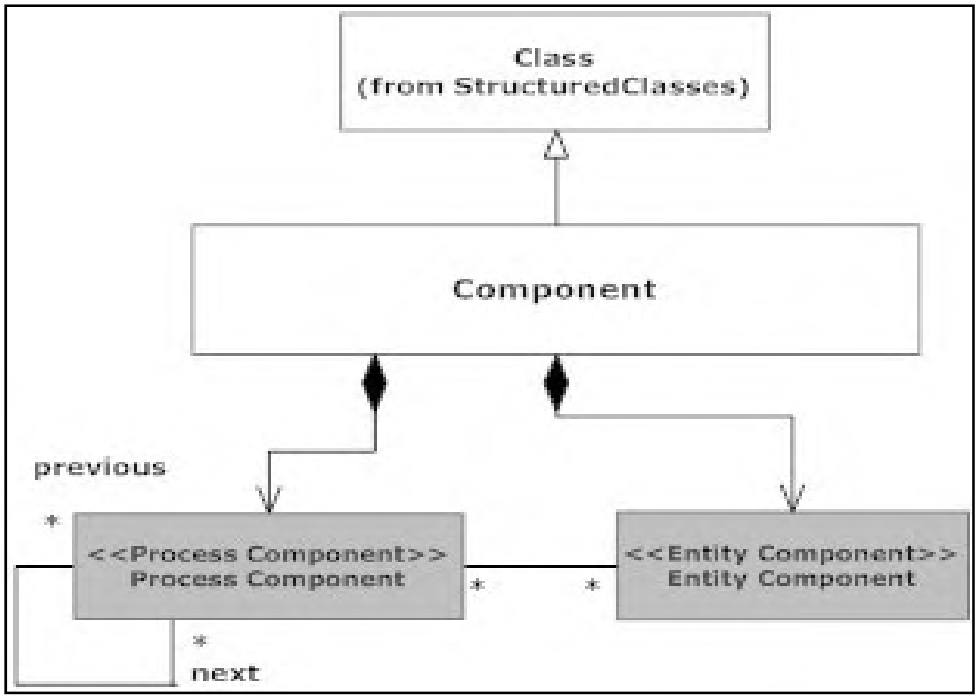


Figure 6.2 Le métamodèle de Business Component pattern

6.2.5 Assemblage du modèle de composants d'affaires (BCM)

Après avoir identifié les composants-processus et leur enchaînement puis les composants-entités, nous pouvons relier les composants-processus entre eux, selon l'enchaînement des Activities du BPM. Nous pouvons ensuite relier chaque composant-entité à un composant-processus avec lequel il interagit. La Figure 6.3 présente le métamodèle du profil de modèle UML des composants d'affaires basé sur le métamodèle du diagramme de composants UML2.

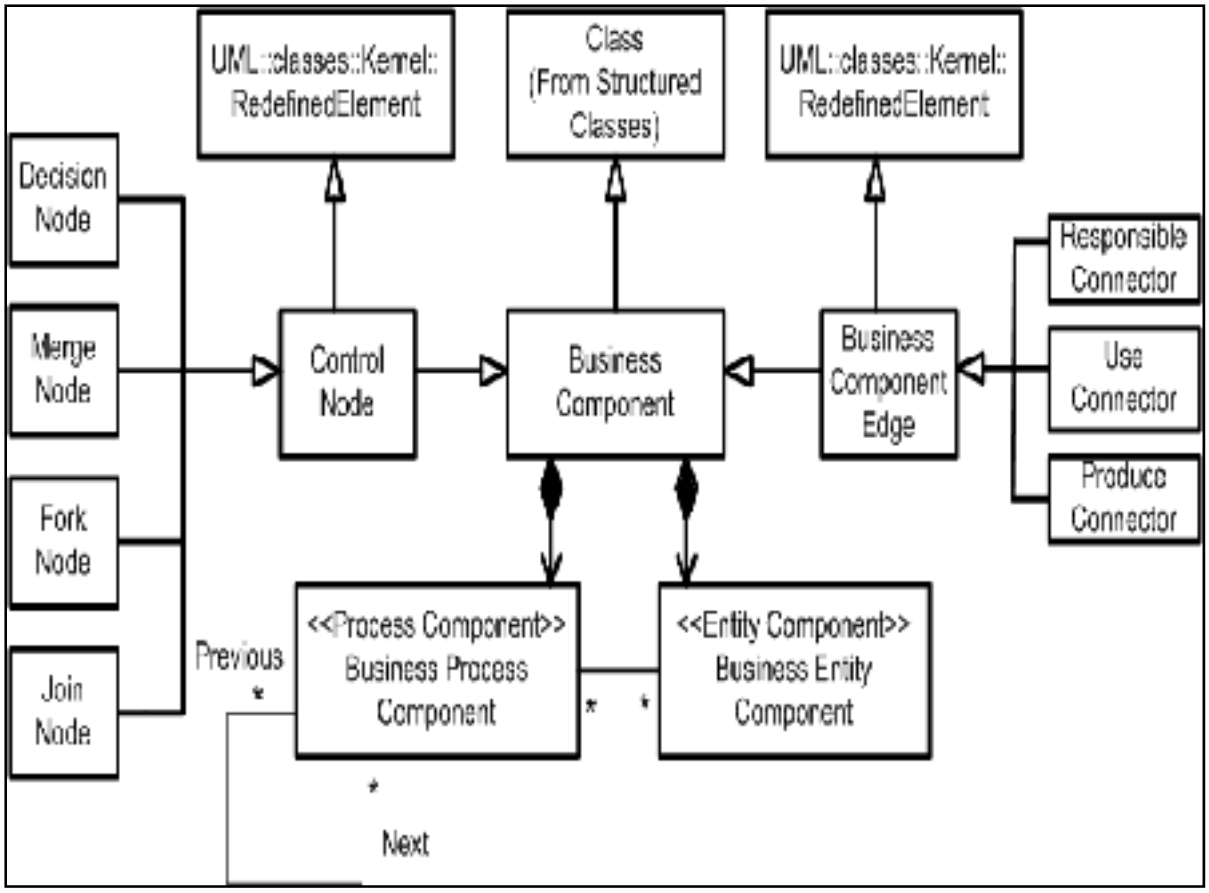


Figure 6.3 Métamodèle du profil de modèle UML de composants d'affaires

6.3 Proposition d'un nouveau diagramme UML des composants d'affaires

Le profil BCM permet de doter l'UML d'un nouveau diagramme spécifique à la modélisation des composants d'affaires : le diagramme des composants d'affaires.

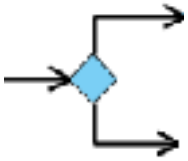
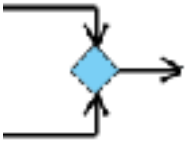
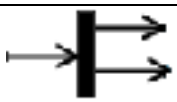
Le profil BCM est une adaptation d'UML basée sur le MOF. Il utilise les mécanismes d'extension offerts par l'UML tout en respectant le MOF. Le BCM est un diagramme UML qui s'inspire à la fois des concepts du diagramme d'activités en terme de comportement dynamique et des concepts du diagramme de composants en terme de l'organisation du système du point de vue des modules d'affaires. Ce profil BCM fournit deux stéréotypes : *Business Process* et *Business Entity*. Le *Business Process* fait référence au composant

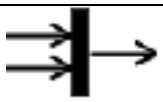
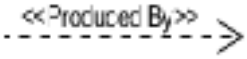
dynamique du modèle BCM et le *Business Entity* fait référence au composant statique du modèle BCM.

Nous avons enrichi le diagramme UML de BCM des nœuds de contrôle, des connecteurs et des attributs. Ils sont détaillés dans le Tableau 6.1.

Il est important de mentionner que le diagramme UML de BCM est conforme au métamodèle d'UML, le MOF.

Tableau 6.1 Éléments du diagramme UML de composant d'affaires

Symbole de l'élément	Nom de l'élément	Explications
	Business Process Component	Composant d'affaires dynamique construit à partir d'un seul *Activity* du BPM.
	Business Entity Component	Composant d'affaires statique construit à partir d'un ou plusieurs Objets *Resources*, *Results* et *Actor* du BPM.
	Initial Node	Représente le nœud initial du modèle des composants d'affaires.
	Final Node	Représente le nœud final du modèle des composants d'affaires
	Decision Node	Représente un point de choix entre les composants d'affaires processus. Il ne s'applique pas aux composants d'affaires entités.
	Merge Node	Fournit un moyen de réunir les flux qui proviennent d'un ou plusieurs nœuds de décision en amont. Il s'applique uniquement aux composants d'affaires processus.
	Fork Node	Permet uniquement à un nombre de composants d'affaires processus de se créer.

	Join Node	Permet de réunir uniquement les composants d'affaires processus.
	Used By Connector	Permet de connecter uniquement un composant d'affaires entité de type *Resources* au composant d'affaires processus
	Produce Connector	Permet de connecter uniquement un composant d'affaires entité de type *Results* au composant d'affaires processus
	Responsible Connector	Permet de connecter uniquement un composant d'affaires entité de type *Actor* au composant d'affaires processus
	Simple Connector	Permet de connecter uniquement les composants d'affaires processus
Id	Identification of Business Component	Numéro ou code d'identification d'un composant d'affaires processus ou d'un composant d'affaires entité
Type : (Role, Resources, Results)	Type of Business Entity Component	Type de composant d'affaires entité : type *Role*, type *Resources* ou Type *Results*

6.4 Règles de cohérence intra-BCM

Sur la base des règles de cohérence du patron EBP mentionnées dans la section 5.1.3, nous avons identifié trois règles intra-BCM. Ces règles expriment la cohérence et les relations entre les deux composants d'affaires *Business Process* et *Business Entity* au sein d'un BCM. Les connecteurs ***Used by***, ***Produce*** et ***Responsible*** représentent l'application directe de ces trois règles.

RC-BCM1 : un composant d'affaires *Business Entity* de type *Actor* est responsable du composant d'affaires *Business Process*. En d'autres termes, la responsabilité de l'*Actor* se définit par l'exécution du composant d'affaires *Business Process* sous la responsabilité de l'acteur.

RC-BCM2 : un composant d'affaires *Business Entity* de type *Resources* est utilisé par le composant d'affaires *Business Process*.

RC-BCM3 : un composant d'affaires *Business Process* produit (comme résultat de son exécution) un composant d'affaires de type *Results*.

6.5 Transformation du BPM en BCM

La transformation du BPM en BCM sera basée sur la correspondance entre les éléments dynamiques et les éléments statiques de chacun des deux modèles. En d'autres termes, les éléments dynamiques dans le BPM seront transformés en éléments dynamiques dans le BCM, de même que les éléments statiques. En effet, la correspondance entre les deux modèles, le BPM et le BCM, se réalise selon la correspondance entre le patron EBP (EBP pattern) et le patron de composants d'affaires (BC pattern) (Figure 6.4).

Les règles de transformations suivantes assurent la correspondance entre ces deux patrons :

R1 : L'élément ***Activity*** de l'EBP pattern se transforme en l'élément ***Business Process component***;
R2 : L'élément ***Resources*** de l'EBP pattern se transforme en l'élément ***Business Entity Component;***
R3 : L'élément ***Results*** de l'EBP pattern se transforme en l'élément ***Business Entity Component*;**
R4 : L'élément ***Actor*** de l'EBP pattern se transforme en ***Business Entity Component.***

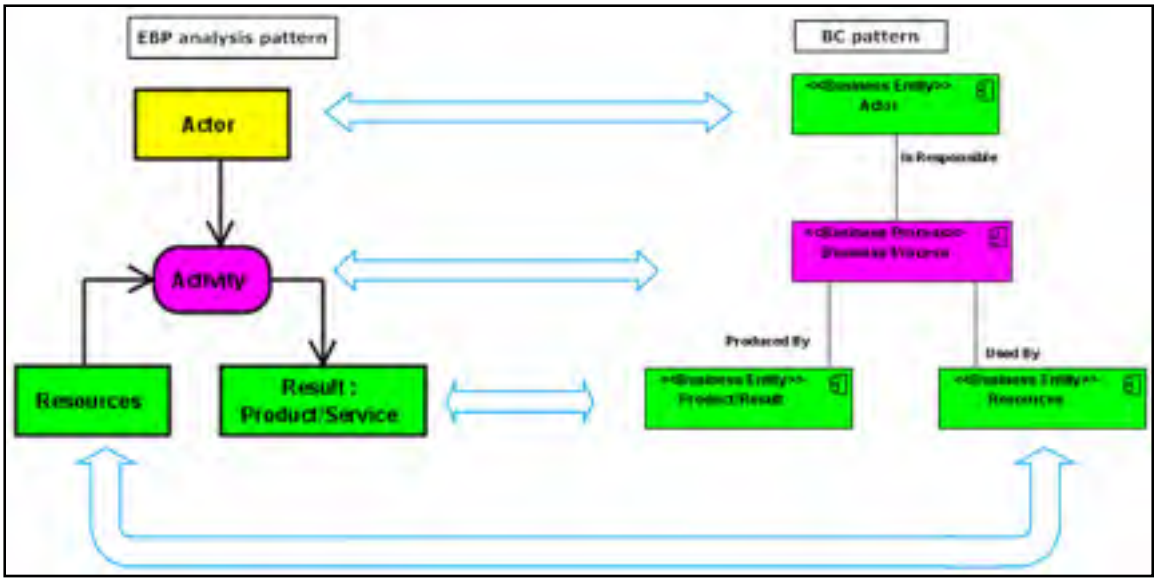


Figure 6.4 Correspondance entre l'EBP pattern et le BC pattern

Le Tableau 6.2 résume la correspondance entre les éléments dynamiques et les éléments statiques de l'EBP pattern et le BC pattern.

Tableau 6.2 Correspondance entre les éléments dynamiques et les éléments statiques de l'EBP et du composant d'affaires

	EBP pattern	BC pattern
Éléments Dynamiques	Activity	Process Component
Éléments Statiques	Actor Resources Results	Entity Component

6.6 Procédure de transformation du BPM au BCM au niveau modèle

La procédure de la transformation du BPM au BCM se compose en deux étapes : la première consiste en l'application des règles de transformation automatiques et la seconde, en l'application des règles de transformation manuelles. Il est à noter que cette procédure de transformation se réalise au niveau modèle.

Étape 1 : règles de transformation automatiques

RA1 : Chaque élément *Activity* du BPM se transforme en un élément *Process Component* du BCM;
RA2 : Chaque élément *Actor* du BPM se transforme en un élément *Entity Component* du BCM;
RA3 : Chaque élément *Object Node* de type *Ressources* du BPM se transforme en un élément *Entity Component* du BCM;
RA4 : Chaque élément *Object Node* de type *Products/Results* du BPM se transforme en un élément *Entity Component* du BCM;
RA5 : Les *Process Component* sont reliés entre eux selon l'enchaînement des éléments *Activities* du BPM;
RA6 : Les **Entity Process** sont reliés respectivement aux **Process Component** selon leurs liaisons aux éléments **Actitivies** du BPM;
RA7 : Le nom de chaque élément du BPM doit être le même dans le BCM ou au moins un nom dérivé pour garder une nomenclature cohérente. Par exemple, si le nom de l'*Activity* dans le BPM est ''Reserve'', le nom correspondant dans l'élément *Process Component* doit être ''Reserve'' ou ''Reservation'';
RA8 : les pré-conditions et les post-conditions de l'élément *Activity* du BPM seront les mêmes pré-conditions et post-conditions de l'élément *Process Component* du BCM;
RA9 : les nœuds de contrôle, le nœud initial et le nœud final seront les mêmes dans le BCM selon leur ordre d'apparition dans le BCM.

Le Tableau 6.3 résume les règles de transformation automatique du BPM en BCM en mettant en correspondance les éléments du BPM et les éléments du BCM.

Tableau 6.3 Correspondance entre les éléments du BPM et les éléments du BCM selon les règles de transformation automatiques

Business Process Model Concepts	**Business Component Model Concepts**
Actor (swimlane)	*Entity Component*
Activity	*Process Component*
Resources (ObjecNodet)	*Entity Component*
Results (ObjectNode)	*Entity Component*
Transition between Activity	*Transition between Process Component*
Guard on Transition	*Constraint on association*
Start Node	*Start Node*
End Node	*End Node*
Decision Node	*Decision Node*
Join Node	*Join Node*
Fork Node	*Fork Node*
Merge Node	*Merge Node*

Étape 2 : Règles de transformation manuelles

Pour compléter la conception du BCM, il existe une seule règle de type manuelle. Parfois, pour des raisons de contraintes conceptuelles, il se trouve que la granularité d'un *Entity Component* est assez élémentaire et ne permet pas dans ce cas-ci de former un composant d'affaires de type *Entity*.

Prenons le cas d'une carte de crédit : Une carte de crédit est un objet de type ressources dans le BPM. Conceptuellement, cet objet est physiquement la propriété d'un certain acteur. Si nous le transformons selon la règle de transformation RA3, nous obtiendrons un *Entity*

Component dans le BCM. Or, une carte de crédit ne peut exister seule et représente un *Entity Component* de granularité élémentaire.

Dans ce genre de situation où l'architecte du logiciel est maître de la situation, il est préférable de regrouper les *Entity Component* dans un seul *Package*. C'est une façon de dire qu'il existe un lien fort entre ces *Entity Components*.

Dans certains cas, il peut exister un lien entre deux *Entity Components* comme dans la situation ou l'*Entity Component* de type Client est propriétaire et utilisateur de l'*Entity Component* de type Carte de Crédit.

Dans d'autres cas, il se peut qu'un *Entity Component* de granularité faible existe seul comme dans le cas d'un *Entity Component* de type Contrat. Cette situation représente une limite de notre recherche et nous confronte à la situation suivante : théoriquement la granularité d'un composant d'affaires est élevée alors que dans certains cas pratiques, il se trouve qu'un *Entity Component* est de granularité faible.

Voici donc la règle RA10 à appliquer dans le cas de l'existence d'un *Entity Component* de granularité faible. Cette règle est seulement valide lorsqu'il existe un lien entre deux *Entity Components* ou plus. Le lien entre les deux composants d'affaires permet de former un *Package* d'affaires.

RA10 :

- Mettre les *Entity Components* dans un *Package*;
- Le nom du *Package* doit porter le nom de l'*Entity Component* principal. Celui-ci inclut, utilise ou est responsable des autres *Entity Components*;
- Les *Entity Components* de ce *Package* doivent être reliés à l'*Entity Component* du même *Package* par la relation *Used By*;
- L'*Entity Component* principal doit être relié par une association *Is Responsible* avec le *Process Component* avec lequel il interagit directement.

6.7 Transformation du BPM au BCM au niveau des métamodèles selon les profils proposés

Le Tableau 6.4 présente la correspondance, au niveau métamodèle, entre les éléments du BPM et les éléments du BCM. Cette correspondance est basée sur les règles de transformation automatiques, la correspondance entre les éléments du BPM et les éléments du BCM au niveau modèle (Tableau 6.3), le métamodèle du profil UML2 du BPM et le métamodèle du profil UML2 du BCM.

Tableau 6.4 Correspondance entre les éléments du BPM et les éléments du BCM au niveau métamodèle

Source (BPM) : **Profil UML2 du BPM**	**Destination (BCM) :** **Profil UML2 du BCM**
Activity	Business Process Component
Resources	Business Entity Component
Products/Results	Business Entity Component
Actor	Business Entity Component
Merge Node	Merge Node
Decision Node	Decision Node
Fork Node	Fork Node
Join Node	Join Node
Guard on Transition	Business Component Edge
Start Node	Start Node
End Node	End Node

6.8 Synthèse

Ce chapitre présente le modèle des composants d'affaires BCM, une composante du CIM permettant de représenter les processus d'affaires élémentaires (EBP) à un autre niveau de granularité d'affaires. C'est un modèle intermédiaire qui agit comme un connecteur entre le modèle des processus d'affaires, le modèle des cas d'utilisation, le modèle de classe et le

modèle de composants système. Plus particulièrement, la démarche de conception du BCM et la méthode permettant de transformer le BPM en BCM sont présentées.

La conception du BCM s'appuie sur la catégorisation des composants d'affaires en terme de composant-processus et de composant-entité. Un nouveau diagramme UML ainsi que son métamodèle permettant de modèliser le BCM ont été proposés.

Afin de permettre la génération du BCM à partir du BPM, dix règles de transformations automatiques et une règle de transformation manuelle, prenant en entrée le BPM et en sortie le BCM, ont été proposées.

CHAPITRE 7

TRANSFORMATION DU MODÈLE DES PROCESSUS D'AFFAIRES AU MODÈLE DES CAS D'UTILISATION

Ce chapitre présente une partie de la phase 4 de ce projet de recherche, soit la technique permettant la transformation du modèle des processus d'affaires (BPM) vers le modèle des cas d'utilisation (UCM).

7.1 Du BPM à l'UCM : Règles de transformation

Dans le BPM, il existe trois principaux types d'éléments soit l'acteur, l'activité et l'objet. Dans le modèle des cas d'utilisation, il existe deux principaux types d'éléments, soit l'acteur et le cas d'utilisation.

Nous transformons l'acteur du BPM en acteur du modèle des cas d'utilisation. Selon la définition, un acteur joue un ensemble cohérent de rôles que les utilisateurs incarnent au moment de leurs interactions avec le système. En revanche, la sémantique d'UML ne précise aucune contrainte quant à la considération de ce qui est exactement un ensemble cohérent de rôles. Nous assumons dans ce cas que toute cohérence est acceptée. Cependant, un rôle est défini dans le BPM comme un groupe d'entités qui a les mêmes droits et obligations à l'égard de l'exécution d'une activité ou d'un groupe d'activités. En résumé, un acteur du BPM définit un ensemble cohérent de rôles que les utilisateurs incarnent au moment de l'exécution des activités.

Une activité du BPM définit un comportement décrit par une séquence organisée d'unités dont les éléments simples sont les actions. Sous la responsabilité d'un acteur, cette activité consomme des ressources et produit un résultat. Par ailleurs, un cas d'utilisation décrit l'interaction entre un acteur et le système. L'interaction est une séquence d'actions qui manipule un ensemble d'objets de domaines et produit un résultat tangible pour l'acteur. Par

conséquent, et par analogie, la définition d'une activité du BPM se rapproche de celle d'un cas d'utilisation. Nous pouvons déduire qu'une activité du BPM se transforme en un cas d'utilisation au moment où nous introduisons le système comme acteur.

En revanche, selon notre catégorisation de l'activité du BPM en activité supportée, manuelle et automatique, seules les deux catégories supportée et automatique remplissent ce rapprochement. Une activité BPM manuelle ne peut jamais interagir avec un acteur de type système. Donc, cette activité ne sera jamais prise en compte lors du développement du système bien qu'elle fasse partie du modèle des processus d'affaires et du système d'information. D'ailleurs, un cas d'utilisation est une interaction entre un acteur et un système.

En effet, le modèle des processus d'affaires a pour finalité de cartographier l'ensemble des processus d'affaires de l'entreprise. Or, le modèle des cas d'utilisation recense l'ensemble des fonctionnalités du système que nous désirons développer.

En résumé, pour transformer un BPM en un modèle de cas d'utilisation UCM, nous recommandons l'application des règles de transformations suivantes :

R1 : chaque activité (Activity) supportée du BPM sera transformée en un cas d'utilisation dans l'UCM;
R2 : chaque activité (Activity) automatique du BPM sera transformée en un cas d'utilisation dans l'UCM;
R3 : chaque acteur (Actor) du BPM sera transformé en un ou plusieurs acteurs (Actors) dans l'UCM. Il sera associé au cas d'utilisation transformé à partir de l'*Activity* dont l'acteur est responsable de son exécution dans le BPM;
R4 : Les objets ressources (Resources) et les objets résultats (Results) de l'activité du BPM seront transformés en des objets dans l'UCM. Ils seront manipulés par le cas d'utilisation généré à partir de l'activité du BPM et ce, en utilisant la règle de transformation **R1** ou **R2**;

R5 : Tous les nœuds de contrôles du BPM seront transformés en des nœuds de contrôle de l'UCM selon leur ordre d'apparition dans le BPM.

La Figure 7.1 illustre les quatre règles R1, R2, R3 et R4 de transformation du BPM à l'UCM. La règle R5 n'est pas illustrée dans cette figure pour des raisons de simplification.

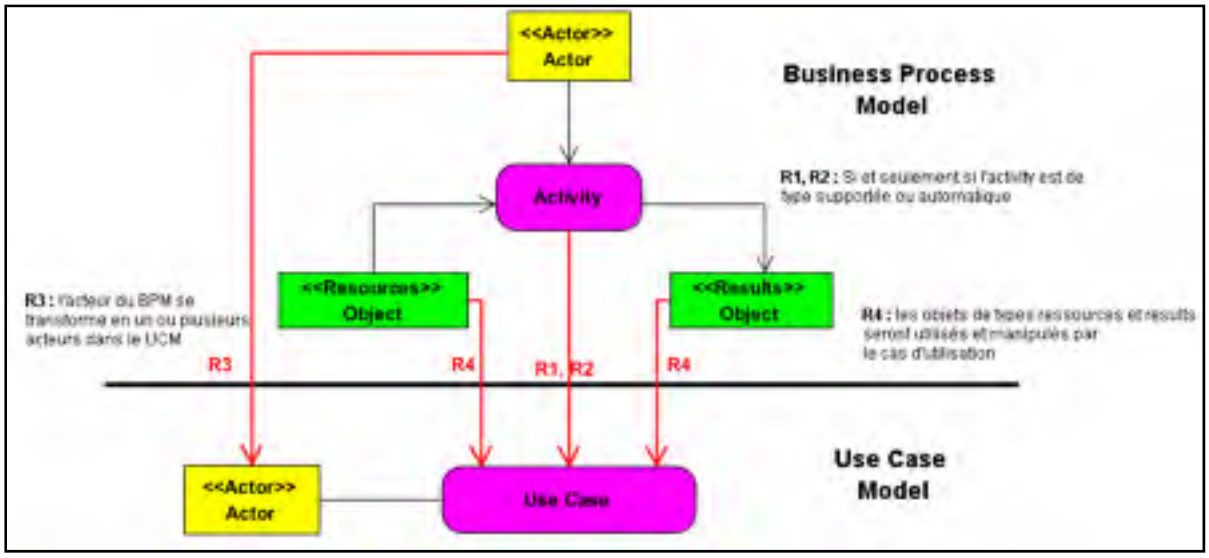


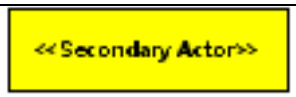
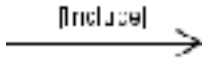
Figure 7.1 Règles de transformation du BPM à l'UCM

7.2 Vers un nouveau format de présentation du modèle de cas d'utilisation

Habituellement, l'UCM est modélisé à l'aide du diagramme de cas d'utilisation UML. Ce diagramme permet de donner une vision globale du comportement d'un système logiciel et est largement accepté par la communauté logicielle. En revanche, selon notre démarche, nous proposons la génération de l'UCM à partir du BPM : celui-ci contient des objets, des acteurs et des activités. Il est donc normal pour nous de prendre en compte les objets dans l'UCM. De ce fait, nous proposons une nouvelle technique qui exploite la force du diagramme d'activité UML2 pour concevoir l'UCM.

De ce fait, nous avons introduit des nouveaux éléments dans le diagramme d'activité UML2 afin qu'il puisse supporter la modélisation de l'UCM. Ils sont expliqués dans le Tableau 7.1.

Tableau 7.1 Les nouveaux éléments de l'UCM modélisé par le diagramme d'activités

Éléments	Explications	Légendes
Acteur secondaire	Le rôle joué par un acteur humain ou un acteur de type système. C'est un élément de type objet stéréotypé *Secondary Actor*.	
Relation d'inclusion	L'instance du cas d'utilisation source comprend également le comportement décrit par le cas d'utilisation destination.	
Relation d'extension	Le cas d'utilisation source étend le comportement du cas d'utilisation destination.	
Relation d'héritage	Un cas d'utilisation hérite du même rôle joué par un autre cas d'utilisation. La relation d'héritage est aussi applicable quand un acteur hérite du comportement d'un autre acteur.	

Cependant, la relation d'héritage entre les acteurs principaux de l'UCM devient difficilement applicable dans ce cas-ci. Cela est dû au choix fait pour représenter les acteurs principaux à l'aide des *Swimlanes*. Un *Swimlane* en UML n'a pas de sémantique significative (OMG, 2009-02-02), il indique seulement un certain regroupement quant à l'organisation des actions et des activités. Toutefois, il est tout à fait possible d'utiliser l'héritage entre les acteurs principaux si nous changeons le *Swimlane* qui représente l'acteur principal par un objet qui porte le stéréotype Principal *Actor*. Dans ce cas-ci, les *Swimlanes* ne seront pas utilisés et nous aurons deux possibilités de concevoir un UCM.

7.2.1 Deux catégories de cas d'utilisation

L'utilité des cas d'utilisation varie selon leur pertinence. Pour atteindre l'objectif d'affaires, certains sont d'une importance plus élevée que d'autres. Les plus importants, méritent d'être détaillés davantage. Les autres, qui sont d'une importance relativement faible, ne nécessitent pas d'être détaillés par un cas d'utilisation.

Les auteurs dans (Kurt Bittner et Ian Spence, 2002) recommande de s'intéresser uniquement aux cas d'utilisation dont la finalité est de concrétiser le but de l'acteur comme une réservation de voiture, un retrait d'argent, une commande de produit, etc. Par contre, les cas d'utilisation de type CRUD (Create, Read, Update, Delete) qui manipule directement un objet ne représentent pas une grande importance quant à l'activité du développement de système logiciel. Généralement, ils sont basiques avec des actions redondantes. Ils se réduisent à entrer, valider et modifier une donnée. Ils sont rarement reliés aux buts de l'acteur car ils représentent souvent un ensemble de différents buts à partir d'une variété de processus et de contextes d'affaires. Ils sont généralement longs à rédiger et souvent associés à de nombreux acteurs.

En effet, selon notre BPM nous constatons que les objets de type *Resources* et *Results* appartiennent à cette catégorie de cas d'utilisation CRUD. Ceci nous amène à les catégoriser et les distinguer des cas d'utilisation issus de l'activité du BPM. Il est donc important pour nous de mentionner cette différence :

- Les cas d'utilisation transactionnels sont les cas d'utilisation qui nous intéressent dans notre recherche. Ils sont les résultats directs de la transformation de l'élément *Activity* du BPM à l'UCM.
- Les cas d'utilisation CRUD sont les objets de type *Resources* et *Results* de l'UCM.

Il est à noter que dans cette recherche, nous nous intéressons uniquement aux cas d'utilisation transactionnels.

7.2.2 Vers un métamodèle du profil de l'UCM

Dans le même ordre d'idée que le profil BPM, nous profitons de la flexibilité d'UML pour proposer un profil adapté à la modélisation de l'UCM à l'aide du diagramme d'activités UML2. L'utilisation de ce diagramme, à la fois pour concevoir le BPM et l'UCM contribue à préserver une traçabilité entre les éléments des deux modèles. L'utilisation d'un seul diagramme réduit à la fois la courbe d'apprentissage et contribue à focaliser sur l'activité de

l'analyse à l'égard de l'apprentissage d'une nouvelle technologie de modélisation. La Figure 7.2 illustre le métamodèle du profil UCM.

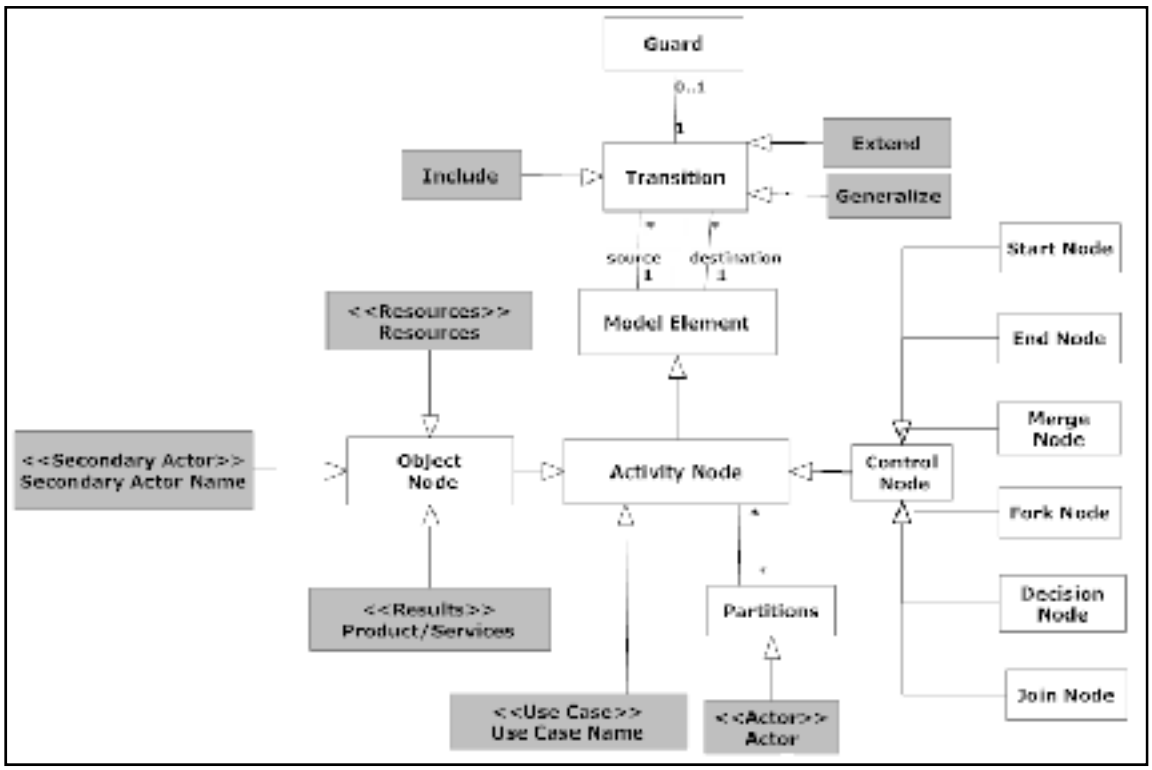


Figure 7.2 Métamodèle du profil du modèle des cas d'utilisation (UCM)

Comme l'illustre la Figure 7.2, les nouveaux éléments qui ont été ajoutés au profil UCM sont en couleur grise. En voici une courte description :

- *Extend*, *Include* et *Generalize* sont des éléments qui héritent le l'élément Transition, ils sont orientés (leur direction a un sens) et stéréotypés;
- Les objets de type *Resources* et *Results* sont du même type que ceux du profil BPM;
- *Secondary Actor* est l'acteur secondaire du cas d'utilisation de type objet. Il hérite de l'élément *Object Node* et prend le stéréotype *Secondary Actor* pour se différencier des autres objets;
- *Use Case* est le cas d'utilisation. Il hérite de l'élément *Activity Node* et prend le stéréotype *Use Case*;

- *Actor* est l'acteur principal d'un cas d'utilisation. Il hérite de l'élément *Partition* et prend le stéréotype *Actor*.

Il est à préciser que si nous optons pour construire l'UCM sans l'utilisation des partitions (Swimalne) l'élément *Actor* dans le profil UCM n'héritera plus de l'élément *Partition* mais plutôt de l'élément *Object Node*.

7.3 Synthèse

Ce chapitre propose un nouveau format de modèle de cas d'utilisation (UCM). Celui-ci, à la différence du diagramme de cas d'utilisation UML, est basé sur le métamodèle du diagramme d'activité UML2. Il introduit de nouveaux éléments, notamment, la notion de l'objet.

Il est aussi proposé qu'il est possible de générer l'UCM à partir du BPM. L'identification des activités supportées et automatiques du BPM ainsi que l'application des cinq règles de transformation réalisent cette opération. De plus, un métamodèle du profil de l'UCM a été proposé.

CHAPITRE 8

DOCUMENTATION DES CAS D'UTILISATION

8.1 Introduction

Ce chapitre présente une partie de la phase 4 de ce projet de recherche, soit la documentation textuelle d'un cas d'utilisation en utilisant un gabarit.

La rédaction des cas d'utilisation consiste souvent en l'emploi d'un gabarit qui permet de spécifier l'interaction entre l'acteur et le système à l'aide du langage naturel. L'ambiguïté et l'incomplétude sont des problèmes typiques et récurrents. Dans la plupart des cas, la conception et le test du produit final ne répondent pas aux besoins des clients, et empêchent l'utilisation des outils d'automatisation. La plupart des solutions proposées dans la littérature préconisent de rédiger directement les cas d'utilisation sous forme de langage formel; d'autres proposent des mécanismes de traduction du langage naturel vers des langages formels (Butler, Grogono et Khendek, 1997; Grieskamp et Lepper, 2000; Sengupta et Bhattacharya, 2006). Ceux-ci offrent de nombreux moyens de s'assurer de la cohérence des exigences. Cependant il n'est pas envisageable de disposer de personnel compétent dans les langages formels, surtout dans le contexte des applications logicielles de gestion dont les spécialistes ne sont pas toujours des informaticiens. Par ailleurs, la plupart de gabarits formels sont accompagnés de larges parts de langage naturel, et sont inutilisables sans lui.

Les ambiguïtés dans un cas d'utilisation consistent en des expressions sans références à un contexte particulier, ou à des mots qui ont plus d'un sens dans un contexte particulier. Ces expressions devraient être remplacées par d'autres expressions qui ont une référence définie dans un contexte particulier. Dans notre cas, nous proposons des verbes génériques et prédéfinis dans un patron de communication, qui structurent l'expression de chaque acteur dans un cas d'utilisation. Cette technique nous est utile pour formaliser les cas d'utilisation en vue de faciliter leurs transformations.

L'approche que nous proposons est basée sur un langage naturel semi-contrôlé, qui sert de langage de référence à l'écriture des cas d'utilisation. Ce langage utilise la force des patrons de communication que nous introduirons dans la section 8.4 et des verbes génériques pour exprimer les actions de l'acteur et du système. Un mécanisme est proposé, à l'aide des règles de restriction, pour minimiser l'ensemble des mots utilisés à l'écriture des cas d'utilisation. Le scénario du cas d'utilisation est ainsi rendu écrit avec un minimum de mots. Être capable de minimiser le nombre de mots dans l'élaboration d'un cas d'utilisation permet de réduire le fossé de leur interprétation, et nous paraît être une caractéristique importante de cette approche.

Dans notre recherche, la spécification des cas d'utilisation d'une manière semi-formelle a un intérêt particulier dans la génération du modèle de classe d'analyse. Généralement, les cas d'utilisation spécifiés sous forme d'un modèle structuré (conditions, scénario nominal, post-condition…) améliorent la compréhension humaine. Toutefois, pour qu'ils soient évalués et compréhensibles par des outils logiciels, il faut définir une syntaxe et une sémantique pour chaque expression d'un acteur de cas d'utilisation, ce qui nous emmène à proposer le concept du patron de communication.

L'incomplétude des cas d'utilisation textuels est souvent causée par un manque d'objets, d'acteurs, de transitions ou des relations entre les expressions. Pour minimiser cette ambiguïté et cette incomplétude, nos patrons de communication structureront l'expression de chacun des acteurs dans un cas d'utilisation. À l'aide de la sémantique et de la structure linguistique des patrons de communication, l'analyse et l'interprétation de chaque expression par un outil logiciel seront facilitées.

8.2 Gabarit d'écriture de cas d'utilisation

Le gabarit que nous préconisons prend la forme d'un tableau (Tableau 8.1) qui structure la manière d'écrire un cas d'utilisation. Il se compose de treize principaux éléments. Il n'est pas

nécessaire de discuter en profondeur les onze premiers (Use case Name, Version, Problem Domain, Author, Purpose and Description, Actors, Dependency, Generalization, Trigger events, Preconditions et Postconditions), car ils sont déjà acceptés dans la communauté du génie logiciel et sont fondamentaux dans un canevas de cas d'utilisation. En effet, nous les présentons brièvement, un par un, dans le gabarit suivant – Tableau 8.1.

Tableau 8.1 Gabarit de rédaction des cas d'utilisation

<table>
<tr><td colspan="3">Use case Name : nom du cas d'utilisation</td><td colspan="3">Version : version du Use case</td></tr>
<tr><td colspan="3">Problem Domain : nature du domaine du problème</td><td colspan="3">Author : l'auteur du use case</td></tr>
<tr><td colspan="6">Purpose and Description : objectif, utilité et résultat du cas d'utilisation.</td></tr>
<tr><td>Actors</td><td colspan="5">Primary : acteur primaire qui initie le cas d'utilisation
Role of primary actor : le rôle de l'acteur primaire
Secondary : acteur(s) secondaire(s) qui participe(nt) à l'exécution du use case ou un système externe
Role of secondary actor : le rôle de l'acteur secondaire</td></tr>
<tr><td colspan="6">Dependency : relation d'inclusion et/ou d'extension avec les autres cas d'utilisation.</td></tr>
<tr><td colspan="6">Generalization : relations d'héritage avec les autres cas d'utilisation</td></tr>
<tr><td colspan="6">Trigger events : éléments déclencheurs du cas d'utilisation</td></tr>
<tr><td colspan="6">Preconditions : conditions nécessaires avant l'exécution du cas d'utilisation</td></tr>
<tr><td colspan="6">Postconditions : conditions nécessaires après l'exécution du cas d'utilisation</td></tr>
<tr><td colspan="6">Main Scenario</td></tr>
<tr><td>Step</td><td>Primary actor</td><td colspan="2">Secondary actor</td><td>System response</td><td>Decision</td></tr>
<tr><td>Numéro de l'étape</td><td>Actor communication pattern (ACP)</td><td colspan="2">Actor communication pattern (ACP)</td><td>System communication pattern (SCP)</td><td>Règle de décision</td></tr>
<tr><td colspan="6">Alternate Scenarios</td></tr>
<tr><td>Alternate Step</td><td>Alternate primary actor</td><td colspan="2">Alternate secondary actor</td><td>Alternate system response</td><td>Alternate decision</td></tr>
<tr><td>Numéro de l'étape alternative</td><td>Actor communication pattern (ACP)</td><td colspan="2">Actor communication pattern (ACP)</td><td>System communication pattern (SCP)</td><td>Règle de décision</td></tr>
</table>

Les deux éléments qui nous intéressent plus particulièrement sont *Main Scenario* (scénario principal d'un cas d'utilisation) et *Alternate Scenario (*scénario alternatif d'un cas d'utilisation).

Scénario principal (Main Scenario)

Le scénario principal d'un cas d'utilisation décrit une succession d'étapes d'acteurs et de systèmes qui est réussie sans erreurs. Il ne contient aucune condition, ni branchement alternatif (Cockburn, 2000; Larman, 2004). Il est donc recommandé de décrire séparément les flux alternatifs.

Dans notre approche, le scénario principal est composé d'une séquence d'étapes, chaque étape contient une seule expression qui peut exprimer soit le comportement de l'acteur ou le comportement du système. Les comportements de l'acteur et du système sont exprimés à l'aide d'un patron de communication (introduit dans la section 8.4). *L'Actor Communication Pattern* exprime le comportement de l'acteur primaire ou de l'acteur secondaire et le *System Communication Pattern* exprime le comportement du système.

Dans le gabarit proposé au Tableau 8.1, le scénario nominal est enrichi de deux autres éléments, un numéro d'étapes, *Step,* utile pour référencer chaque étape et une décision à prendre, *Decision*, dans le cas où le patron de communication échoue. Cette décision sert à spécifier l'étape à exécuter dans le scénario alternatif lors d'une étape non réussie dans le scénario principal.

Scénario Alternatif (Alternate Scenario)

L'existence d'une étape du scénario alternatif dépend toujours d'une condition ''Decision'' survenue lors d'une prise de décision pendant l'exécution d'une étape du scénario. Cette dernière peut être soit une étape du scénario nominal ou une étape du scénario alternatif. Les

conditions de branchement sont précédées par le mot clé *If* et sont reliées soit à *Main Scenario* ou *Alternate Scenario*.

8.3 Types d'interaction dans un cas d'utilisation

Par définition, un cas d'utilisation est une séquence d'actions qui décrit l'interaction entre un acteur et le système. D'une manière générale, l'acteur qu'il soit primaire ou secondaire, envoie une requête au système puis celui-ci déclenche une réponse en guise de résultat à la stimulation de l'acteur. En effet, un acteur ne peut pas exécuter deux étapes successives sans stimuler le système, mais le système peut effectuer un traitement qui pourra se subdiviser en deux actions successives mais distinctes. Cette situation se produit dans plusieurs circonstances notamment dans le cas où le système valide une donnée après avoir vérifié sa cohérence, et dans le cas où le système change l'état d'un de ses objets et envoie un résultat à l'acteur.

Dans le Tableau 8.2, nous résumons les cinq différentes interactions possibles dans un cas d'utilisation. Les trois premières sont réutilisées à partir des travaux de Cockburn (Cockburn, 2000), la quatrième et la cinquième font partie de notre proposition. Cette catégorisation d'interactions nous est utile afin d'élaborer le gabarit du cas d'utilisation proposé au Tableau 8.1.

Tableau 8.2 Les interactions possibles dans un cas d'utilisation textuel

	Type d'interaction	**Explications**
1	Primary Actor → System	L'acteur insère une donnée ou sélectionne un objet dans le système.
2	System → System	Le système crée, met à jour, lit, ou détruit une donnée ou un objet.
3	System → Primary Actor	Le système envoie un résultat en guise de réponse à l'acteur primaire.
4	System → Secondary Actor	Le système envoie une requête à l'acteur secondaire.
5	Secondary Actor → System	L'acteur secondaire envoie un résultat au système en guise de réponse.

8.4 Patrons de communication

Pour faciliter l'écriture des cas d'utilisation et minimiser le nombre de leurs interprétations, Cockburn (Cockburn, 2000) préconise l'utilisation de phrases courtes, généralement une seule phrase par étape. Dans la même lignée d'idées, Rolland *et al.,* (Rolland Colette et Achour Camille Ben, 1998) ont spécifié sur la base des travaux de Chomsky (Noam Chomsky, 1971) une grammaire définie par un patron sémantique et des structures linguistiques pour mieux contrôler la façon d'écrire un cas d'utilisation. Une structure plus stricte baptisée 'patron de communication' a été proposée par (Matthias Riebisch et Michael Hübner, 2008). Cette structure est présentée comme suit :

Communication (V) [Agent; Object; Source; Destination]

Elle peut être instanciée comme suit :

Communication (select) [Agent: 'the librarian'; Object: the book'; Source: 'the librarian'; Destination: 'in the list of available books']

En effet, nous nous basons sur ce patron de communication pour proposer une structure linguistique plus stricte à l'écriture des cas d'utilisation.

8.4.1 Types de patrons de communication

Nous avons identifié deux types de patrons de communication selon le type de l'acteur :

- un patron de communication pour l'acteur primaire et l'acteur secondaire.
- un autre patron de communication pour le système.

Nous les présentons comme suit :

Communication Pattern (A/S) [Action, Data/Object, Destination]

A/S : **A** pour acteur primaire ou secondaire et **S** pour le système;
Action : fait référence à un verbe d'action de l'acteur ou du système;
Data/Object : c'est l'élément sur lequel est appliqué l'action qui peut être soit une donnée ou un objet;
Destination : c'est l'endroit où résident les données ou les objets. Il peut être une collection d'objets ou de données.

Nous pouvons instancier ce patron de la façon suivante :

Communication (A) [Actor: 'The student'; Action: 'select'; Object: 'the book'; Destination: 'in the list of the available book']

Or, dans certains cas, il n'est pas possible d'instancier tous les éléments du patron de communication. L'existence de l'élément *Destination* dépend des éléments *Data* et *Object*.

La différence principale entre les deux patrons de communication réside dans l'élément *Action*. Chaque élément Action des deux patrons pourra être instancié par des verbes génériques et spécifiques à chaque type d'acteur.

Les patrons de communication présentent une syntaxe stricte (Tableau 8.3). Chaque colonne du tableau contient un terme précis. Ces termes font référence à un glossaire de termes prédéfinis. De cette façon, les descriptions sont transformées d'un vocabulaire libre à un autre plus restreint.

.

Tableau 8.3 Structure du patron de communication

Actor/System	Action	Object or Data	Destination

8.4.1.1 Actor communication pattern (ACP)

L'*Actor Communication Pattern* (ACP) est une spécialisation du *Communication Pattern*. Il se définit par les éléments suivants :

Actor : l'acteur primaire ou l'acteur secondaire.

Action : verbe d'action prédéfini.

Object or Data : objet ou attribut (donnée).

Destination : objet, liste des objets ou liste de données.

Le Tableau 8.4 présente un aperçu et un exemple d'instance de l'ACP.

Tableau 8.4 Actor communication pattern (ACP)

ACP	Actor	Action	Object or Data	Destination
Description	Acteur primaire ou secondaire	Verbe d'action	Attribut ou objet	Objet, liste d'objets ou de données
Exemple	Customer	Select	Car	In the car list
	Customer	Insert	Date	Into reservation

Vers des verbes génériques et spécifiques à l'acteur

Après avoir proposé des patrons de communication pour contrôler la structure syntaxique de chaque étape du cas d'utilisation, nous nous sommes retrouvés face à un défi majeur : Comment minimiser le nombre de verbes d'action utilisés dans chaque étape du cas d'utilisation ?

Après une analyse de plusieurs cas d'utilisation issus de différents projets industriels et académiques, il s'avère que deux verbes d'action résument l'ensemble des verbes utilisés par les analystes, soit *Insert* et *Select*. En effet, l'acteur interagit avec le système via un support de communication qui n'est que le clavier dans le domaine des applications de gestion

d'entreprise. Cette interaction se limite à envoyer des instructions au système pour atteindre un certain objectif de l'acteur. Celui-ci, en utilisant le clavier comme moyen de communication, ne peut qu'effectuer deux actions possibles, soit écrire des instructions en utilisant les boutons du clavier, soit sélectionner une donnée ou un objet en utilisant la souris ou le clavier. Cette limitation est non seulement un grand pas vers l'écriture des cas d'utilisation avec un minimum de mots, mais aussi un défi pour contrôler l'interprétation des cas d'utilisation dans tout le cycle de développement logiciel. Voici un exemple qui met en perspective cette limitation :

ACP	**Actor**	**Verb**	**Object(s)**	**Destination**
Exemple	*Customer*	***Select***	*Car*	*In the car liste*

ACP	**Actor**	**Verb**	**Data**	**Destination**
Exemple	*Customer*	***Insert***	*Personal informations*	*Into customer*

Il est aussi recommandé dans notre recherche d'utiliser un minimum de mots pour instancier les éléments *Object*, *Data* et *Destination*. Toutefois, nous ne pouvons pas préconiser des mots génériques pour qualifier ces éléments car ils changent selon le contexte et l'environnement dans lesquels l'application logicielle sera développée.

8.4.1.2 System communication pattern

Le *System Communication Pattern* (SCP) est une spécialisation du *Communication Pattern*. La différence entre l'ACP et le SCP réside au niveau du type de l'acteur. Le SCP se définit par les éléments suivants :

Actor : c'est le système que nous désirons développer.
Action : verbe d'action prédéfini.
Object or Data : objet ou attribut (donnée).
Destination : objet, liste des objets ou liste de données.

Le Tableau 8.5 présente un aperçu et un exemple d'instance du SCP.

Tableau 8.5 System Communication Pattern (SCP)

SCP	**Subject**	**Verb**	**Object(s) or Attribute(s)**	**Destination**
Description	Système	Verbe d'action	Donnée ou objet	Objet, liste d'objets ou liste de données
Exemple	System	Select	Car	In the list of available cars

Vers des verbes génériques et spécifiques à l'acteur System

Notre analyse des cas d'utilisation s'intéresse aussi à minimiser, voir fixer des verbes d'action génériques qui seront réalisables pour n'importe quel cas d'utilisation. En effet, nous avons constaté durant notre exercice d'analyse l'utilisation d'une variété de verbes d'action. Or, il s'avère que maintes parmi eux partagent la même signification. Nous avons donc identifié douze verbes d'action pour exprimer le type d'action de chaque verbe dans le SCP.

Il est important de mentionner que le système dans le cadre d'un cas d'utilisation ne fait que répondre aux stimuli de l'acteur, à savoir l'action de l'insertion ou celle de la sélection. Il peut changer l'état d'un objet, valider une donnée, effectuer une recherche, effectuer un calcul, etc. En fait, les actions du système se résument à manipuler des données et des objets.

Nous avons identifié des verbes d'action qui manipulent des données et d'autres qui manipulent des objets.

8.4.1.3 Verbes d'action manipulant des données dans le SCP

Le Tableau 8.6 présente les cinq verbes d'action identifiés lors de notre analyse des cas d'utilisation.

Tableau 8.6 Les 5 verbes d'action du SCP manipulant une donnée

SCP	System	Action Verb	Data	Destination
		Validate		
		Report		
		Insert		
		Calculate		
		Send		

Nous expliquons l'utilité de chaque verbe d'action que nous avons proposé :

Validate : valider une donnée.

Report : afficher une donnée.

Insert : insérer une donnée (dans une base de données, dans un fichier…).

Calculate : tout calcul ou traitement de données.

Send : envoyer une donnée à un acteur secondaire (peut être un autre système).

8.4.1.4 Verbes d'action manipulant un objet dans le SCP

Le Tableau 8.7 présente les sept verbes d'action identifiés lors de notre analyse des cas d'utilisation.

Tableau 8.7 Les 7 verbes d'action du SCP manipulant un objet

SCP	System	Action verb	Object	Destination
		Create		
		Delete		
		Update		
		Select		
		Search		
		Associate		
		Dissociate		

Nous expliquerons l'utilité de chaque verbe d'action que nous avons proposé :

Create : créer un objet.

Delete : détruire un objet.

Update : mettre à jour l'état d'un objet.

Select : sélectionner un objet.

Search : chercher un objet.

Associate : associer un objet à une collection d'objets.

Dissociate : dissocier un objet d'une collection d'objets.

Le mot-clé AND dans le SCP

La vocation première d'un cas d'utilisation est de décrire des interactions entre acteurs et système à priori séquentiels : nous débutons l'action après une transition que lorsque celle qui la précède est terminée.

Cependant, dans la pratique il est possible de rencontrer des actions concurrentes. Cela ne signifie pas nécessairement que l'implémentation est concurrente, mais plutôt que les actions décrites ne sont pas liées par une dépendance causale : l'ordre d'exécution n'a pas d'incidence sur le comportement global.

Dans le cas d'une commande, l'action ''livrer commande'' ne peut pas débuter sans que les deux actions ''valider paiement'' et établir ''bon de commande'' ne sont pas terminées.

Cette situation, emmène à ajouter un nouvel élément de type mot clé que nous nommons *AND*. Cet élément s'ajoutera uniquement au *System Communication Pattern*.

L'utilisation de *AND* dans le SCP peut s'effectuer de trois manières différentes :

1. Entre deux verbes d'actions (*Action Verb*) qui manipulent soit des objets et/ou des données;
2. Entre deux données (*Data*) manipulées par un verbe d'action (*Action Verb*);
3. Entre deux objets (*Objects*) manipulés par un verbe d'action (*Action Verb*).

L'utilisation du mot clé *AND* n'est pas obligatoire, car il est tout à fait possible d'écrire un cas d'utilisation sans s'en servir.

8.5 Les règles de restriction dans l'écriture d'un cas d'utilisation

Les règles de restriction sont classifiées en deux groupes :

- des restrictions relatives à l'utilisation des patrons de communication, et
- des restrictions relatives au renforcement de la structure du gabarit du cas d'utilisation.

Nous avons assigné un numéro pour chaque règle de restriction. Les onze premières règles de restrictions (R1 → R11) concernent la structure et la sémantique des patrons de communication. Le Tableau 8.8 explique l'utilité de chaque règle en matière de réduction d'ambiguïté et du fossé d'interprétation.

Les six règles de restriction (R12 → R17) viennent se rajouter aux bonnes pratiques de rédaction d'un cas d'utilisation. Des expressions claires et concises ainsi qu'une cohérence au niveau de l'utilisation du gabarit du cas d'utilisation sont mises de l'avant.

Tableau 8.8 Règles de restriction dans l'écriture d'un cas d'utilisation textuel

#	Description	Explication
R1	Un seul verbe d'action par étape et par patron de communication à l'exception de l'utilisation du mot clé *AND*.	Renforce l'enchaînement des actions dans une étape du patron de communication.
R2	Utiliser seulement les verbes d'action préconisés dans le SCP et l'ACP.	Renforce le contrôle de la sémantique des patrons de communication et réduit le fossé dans l'interprétation des verbes d'action.
R3	Seul le temps présent est autorisé pour écrire les patrons de communication.	Décrit ce que fait le système plutôt que ce qu'il devra faire ou a déjà fait. Montre explicitement les objets et les données.
R4	L'ACP et le SCP doivent être en mode actif et non en mode passif.	

R5	Ne pas utiliser des adverbes (ex. very, more) et des verbes modaux (ex. might, must) dans les patrons de communication.	Les verbes modaux et les adverbes portent à confusion et à l'incertitude
R6	Les patrons ACP et SCP doivent être des phrases déclaratives.	Obligatoire pour donner un sens aux cas d'utilisation.
R7	Ne pas utiliser les pronoms (he, she, this) et les mots (système, acteur) dans les patrons de communication. Ils sont déjà présents dans le gabarit du cas d'utilisation.	Les sujets (acteur, système) sont déjà spécifiés dans les colonnes du gabarit du cas d'utilisation. Facilite la transformation et élimine la redondance
R8	Les patrons ACP et SCP doivent être des déclarations simples. Les noms des éléments Objects, Data, et Destination ne doivent pas changer dans l'ensemble des cas d'utilisation.	Renforce la traçabilité et les liens entre les différents éléments des cas d'utilisation. . Réduit l'ambiguïté et facilite la transformation
R9	Ne pas utiliser des expressions négatives dans les patrons de communication.	
R10	Le mot clé *And* ne s'applique pas sur l'ACP.	
R11	Il est permis d'utiliser les mots clés *From*, *Into*, *in the* à gauche de l'élément ''*Destination*'' du patron de communication.	Aide à spécifier le chemin du verbe d'action envers l'élément *Destination*
R12	Utiliser un seul patron ACP par étape de l'acteur.	Renforce l'enchaînement des actions de l'acteur.
R13	Utiliser un seul patron SCP par étape du système.	Renforce l'enchaînement des actions du système.
R14	La communication directe entre l'acteur primaire et l'acteur secondaire est interdite. Elle doit absolument passer à travers le système. En fait, il est interdit d'enchaîner respectivement une étape de l'acteur primaire et une autre de l'acteur secondaire et vice versa.	Respecte la définition d'un cas d'utilisation (seule une interaction entre un acteur et le système est autorisée). Renforce la séquence des événements.
R15	Les étapes du cas d'utilisation doivent se suivre séquentiellement.	
R16	Une décision s'applique seulement pour les étapes du système.	Facilite la transformation. Renforce la cohérence. C'est le système qui valide les décisions.
R17	Instancier l'acteur primaire et l'acteur secondaire par leur rôle respectif dans le gabarit du cas d'utilisation.	Évite la redondance dans les patrons de communication.

8.6 Cas d'utilisation visuel

Écrire des cas d'utilisation sous une forme textuelle n'est pas toujours facile. Cela nécessite une bonne connaissance du domaine d'affaires, une rigueur et un niveau élevé de précision. Parfois, malgré l'existence de ces caractéristiques, le cas d'utilisation sous format textuel reste difficile à interpréter et à partager entre les différents intervenants du cycle de développement logiciel.

Dans cette recherche nous avons jugé opportun de nous servir d'une autre forme de description qui pourra être facilement interprétable, reproductible, réutilisable et même simulable.

Jacobson (Jacobson, Booch et Rumbaugh, 1999) a défini les cas d'utilisation comme : ''*...a sequence of actions, including variants, that the system can perform, and that yield an observable result of value to a particular actor*''. Selon cette définition, un cas d'utilisation consiste en une séquence d'actions selon une certaine logique qui délivre un résultat observable à un acteur donné. Par ailleurs, le diagramme d'activité UML permet de spécifier des actions a priori séquentielles. Il offre ainsi un pouvoir d'expression permettant de représenter l'ensemble des interactions d'un acteur avec le système dans un cas d'utilisation.

Larman et Jacobson (Ivar Jacobson et Pan-Wei Ng, 2005; Larman, 2004) ont recommandé, comme alternative, de détailler les cas d'utilisation par les diagrammes d'activités. En effet, cette technique a été utilisée et testée dans d'autres recherches (Samir Kherraf, Eric Lefebvre et Witold Suryn, 2008).

Nous préconisons donc l'utilisation du diagramme d'activités comme alternative à l'écriture des cas d'utilisation. Par conséquent, il serait possible à la fois d'écrire et de modéliser un cas d'utilisation à l'aide des deux techniques que nous avons proposées, à savoir le gabarit d'écriture des cas d'utilisation et le diagramme d'activités.

Le diagramme d'activités peut véhiculer plus d'informations que le langage naturel : il permet de visualiser les branches de décision et les nœuds de jointure, de faciliter le contrôle de la qualité en identifiant facilement les flux manquants et de simplifier l'activité du test des fonctionnalités du système. En conclusion, utiliser le diagramme d'activités pour modéliser les séquences d'action d'un cas d'utilisation contribue à l'utilisation d'un seul diagramme pour modéliser à la fois les processus d'affaires et les cas d'utilisation.

8.6.1 Profil du métamodèle de cas d'utilisation visuel

Le diagramme d'activités est un formalisme de modélisation permettant de spécifier différentes interactions, à priori séquentielles. Il peut être utilisé de façon plus informelle pour décrire les enchaînements d'activités dans un BPM ou des séquences d'actions dans un cas d'utilisation. En outre, il offre aussi un pouvoir d'expression très proche des langages de programmation objet : spécification des actions de base (déclaration de variables, affectation…), structures de contrôle (conditionnelles, boucles) ainsi que les instructions particulières à la programmation orientée objet (appels d'opérations, exceptions…). Il est donc bien adapté à la spécification détaillée des traitements en phase de réalisation.

Pour profiter pleinement de ce pouvoir de modélisation, il est impératif pour nous de spécifier un profil de diagramme d'activités dédié uniquement à la modélisation des cas d'utilisation.

En se basant sur la définition d'un cas d'utilisation, et en prenant en compte notre gabarit (Tableau 8.1) servant à rédiger le cas d'utilisation textuel, nous proposons à la Figure 8.1 un profil visuel de cas d'utilisation basé sur le metamodèle du diagramme d'activités.

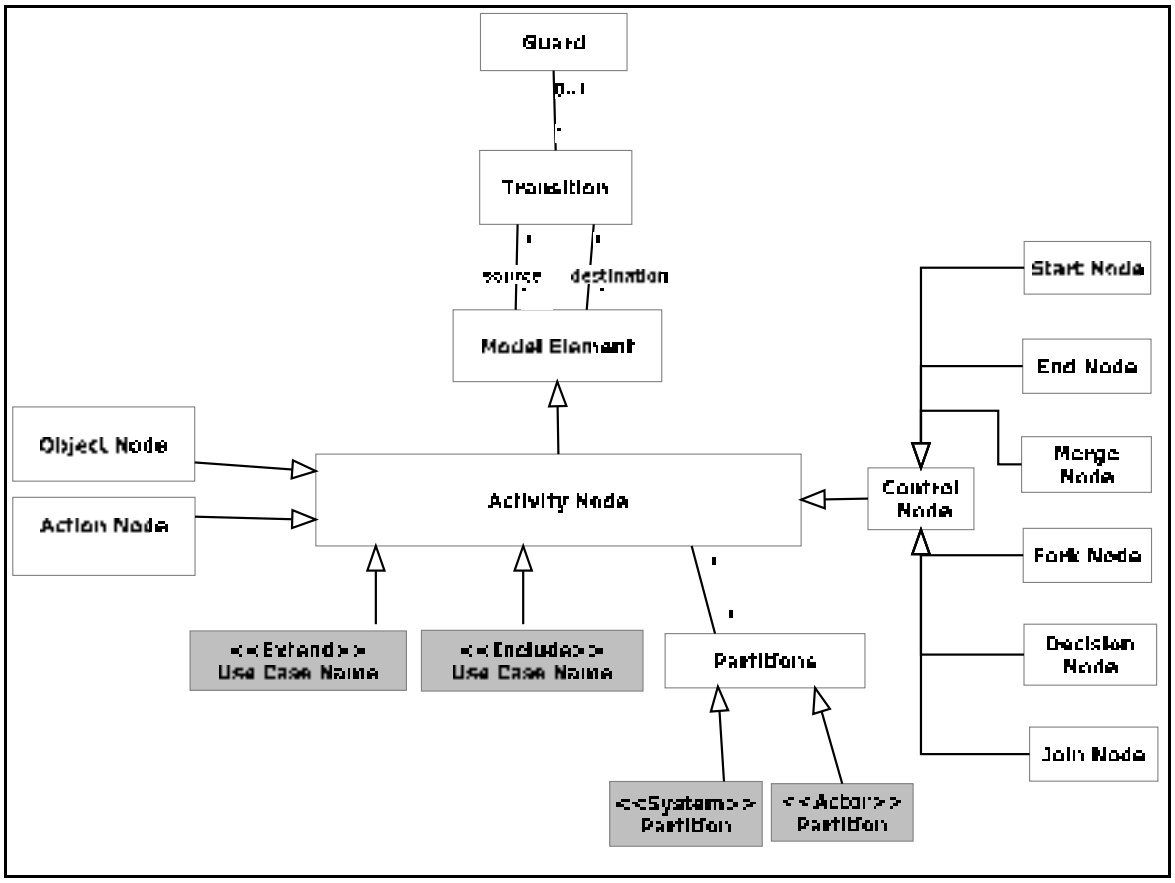


Figure 8.1 Profil visuel de cas d'utilisation

Pour établir une traçabilité bidirectionnelle et faciliter la transformation entre le cas d'utilisation textuel et le cas d'utilisation visuel, nous avons développé, avec quelques variantes, le profil visuel de la même manière que le profil du BPM. En fait, ce profil garde des éléments standards comme les nœuds de contrôle (Start, End, Merge, Fork, Decision et Join), le nœud d'objet, le nœud d'action et la condition de *Guard*. La différence majeure réside dans les éléments suivants :

- **Nœud d'activité** : il est écarté du profil visuel car selon la définition, un cas d'utilisation se constitue uniquement d'une séquence d'actions et non d'une séquence d'activités. Or, nous avons gardé deux variantes du nœud d'activité pour inclure ou étendre un cas d'utilisation par un autre. Les deux stéréotypes réalisent cette possibilité.

- **Partitions** : nous avons créé deux variantes de partitions que nous différencions par les stéréotypes *Actor* et *System*. Ceux-ci servent à modéliser respectivement les acteurs (primaire et secondaire) et le système.
- **Use Case Name** : c'est un nœud d'activité doté du stéréotype ''Extend'' qui fait référence à un autre cas d'utilisation. Il est utilisé dans le cas où un cas d'utilisation étend (Extend) le comportement d'un autre.

8.6.2 Types d'interactions dans le cas d'utilisation visuel

Un cas d'utilisation est une séquence d'actions organisée selon un ordre donné illustrant l'interaction entre l'acteur et le système. Dans le Tableau 8.2, nous avons catégorisé cinq types d'interaction relatifs au cas d'utilisation textuel. Afin de garder une traçabilité pérenne entre les cas d'utilisation textuel et visuel, il est judicieux pour nous de démontrer la possibilité et la manière d'illustrer ces cinq interactions dans le cas d'utilisation visuel.

Dans le profil du cas d'utilisation visuel, il est démontré que les partitions modélisent l'acteur et le système, que les nœuds d'actions modélisent les actions (verbes d'actions dans le patron de communication) et que les nœuds d'objets modélisent les éléments *Object* et *Destination* du patron de communication. Donc, en s'inspirant des interactions acteur-système du cas d'utilisation textuel, il serait possible pour nous d'illustrer, dans le Tableau 8.9, les types d'interactions d'un cas d'utilisation visuel.

Tableau 8.9 Types d'interactions dans un cas d'utilisation visuel

Groupe d'interaction	Type d'interaction	Verbe d'action correspondant
Signal	**Stimulus** : Acteur primaire → Système L'acteur primaire envoie un stimulus au système en sélectionnant un objet ou en insérant une donnée.	Select (object) Insert (data)
	Response : Système → Acteur primaire Le système affiche les données demandées par l'acteur primaire.	Report (data)

Dynamic Flow	Système → Système Le système effectue des traitements sur ses objets et ses données	Validate (data) Report (data) Insert (data) Calculate (data) Create (object) Delete (object) Update (object) Select (object) Seach (object) Associate (object) Dissociate (object)
External Flow	Système → Acteur secondaire Le système envoie un message à l'acteur secondaire.	Send (data)
	Acteur secondaire → System L'acteur secondaire envoie des messages au système.	Send (data)

Pour profiter davantage de la force du diagramme d'activités, nous avons introduit trois groupes d'interactions dans le cas d'utilisation visuel : *Signal*, *Dynamic Flow* et *External Flow*.

1. Le *Signal* se divise en deux types d'interactions :

- Le ***Stimulus*** qui représente un envoi de message de l'acteur primaire envers le système par la sélection d'un objet ou l'insertion d'une donnée.
- Le ***Response*** qui représente la réponse du système à l'acteur primaire par l'affichage des données.

2. Le *Dynamic Flow* se limite à catégoriser l'enchaînement de tous les traitements internes du système, particulièrement ceux qui traitent les données et les objets.

3. L'*External Flow* se concentre sur l'interaction entre le système et les acteurs secondaires. Cette communication se limite à des envois de messages véhiculant des données.

Ensuite, nous avons associé pour chaque type d'interaction du cas d'utilisation visuel, les verbes d'actions génériques qui sont précédemment identifiés dans le cas d'utilisation textuel.

Il est important de noter que l'utilité des éléments de type *Action* du diagramme d'activités est prédéfinie dans les spécifications UML (OMG, 2010a; 2010b; Tim Weilkiens et Bernd Oestereich, 2008). Leurs utilisations se résument comme suit :

- *Action* qui fait appel à un comportement (ex : une autre activité à exécuter).
- *Action* qui fait appel à une opération (C'est une action qui effectue un traitement sur des données).
- *Action* qui envoie et reçoit des signaux.
- *Action* qui crée, supprime et change l'état des objets.

Par analogie, et en comparant les verbes d'actions génériques que nous avons proposé, nous constatons qu'ils partagent la même définition que les actions du diagramme d'activités. Ceux-ci représentent une preuve de plus et renforcent notre proposition.

8.6.3 Règles de contrôle du diagramme de cas d'utilisation visuel

Les règles de contrôle consistent en une série de vérifications de la structure visuelle du cas d'utilisation. Elles sont primordiales pour préserver un certain contrôle entre les différents éléments structurels d'un cas d'utilisation. Les onze règles proposées dans le Tableau 8.10 ont pour vocation de guider l'architecte logiciel à respecter les contraintes de conception afin d'augmenter la qualité du diagramme de cas d'utilisation visuel.

Tableau 8.10 Règles de contrôle d'un cas d'utilisation visuel

#	Description	Explication
R1	Le nœud initial doit être placé dans le *Swimlane* de l'acteur primaire	Dans le domaine de la gestion d'entreprise, c'est l'acteur primaire qui initie le cas d'utilisation.

R2	Le nœud final doit être placé dans le *Swimlane* système	C'est le système qui finalise le cas d'utilisation
R3	Les *Swimlanes* des acteurs primaires et secondaires ne doivent pas contenir de nœuds d'objets.	Un acteur ne fait qu'interagir (envoie des instructions) avec le système via l'interface utilisateur. C'est le système qui traite les objets.
R4	Les *Swimlanes* des acteurs primaires et des acteurs secondaires ne doivent pas contenir respectivement deux actions successives.	L'interaction inter-acteurs et intra-acteurs est interdite. Toute communication entre acteurs et inter-acteur doit passer obligatoirement par le système. Décrire une séquence d'actions d'un acteur relève de la description des interfaces utilisateur et se contredit avec l'objectif du cas d'utilisation.
R5	Il est interdit d'avoir deux actions successives appartenant respectivement au *Swimlane* de l'acteur primaire et au *Swimlane* de l'acteur secondaire, et vice versa.	
R6	Les nœuds de contrôles (decision, fork, join et merge) doivent être uniquement dans le *Swimlane* système.	C'est le système qui est responsable d'exécuter les scénarios alternatifs.
R7	Le *Swimlane* système ne doit pas contenir deux nœuds d'objets successifs	Deux objets ne peuvent jamais communiquer sans une action. Règle de cohérence du diagramme d'activités.
R8	Les attributs seront acheminés via les flux d'objets	Utile pour ne pas surcharger le diagramme d'activité. Les attributs seront incorporés dans les propriétés des flux d'objets et les flux de contrôles.
R9	Dans le cas d'un cas d'utilisation automatique.	Normalement, ce type de cas d'utilisation s'initie par un élément déclencheur de type système. Dans ce cas-ci, nous modélisons ce déclenchement par une précondition attachée au nœud initial du cas d'utilisation.
R10	Chaque nœud de décision sera suivi, plus tard dans le scénario du cas d'utilisation, par un nœud de fusion	Cohérence du diagramme d'activités
R11	Chaque nœud de bifurcation sera suivi, plus tard dans le scénario du cas d'utilisation, par un nœud de jointure	Cohérence du diagramme d'activités

8.7 Règles de transformation du cas d'utilisation textuel en cas d'utilisation visuel

Dans cette section, nous proposons une série de règles de transformation facilitant et cadrant le passage d'un cas d'utilisation textuel en un cas d'utilisation visuel. En effet, pour réussir cette activité de transformation, il est primordial que ce cas d'utilisation textuel respecte toutes les règles d'écriture du gabarit (Tableau 8.1) et ne viole aucune règle de restriction du Tableau 8.8.

Les règles de transformation sont résumées dans le Tableau 8.11 et structurées comme suit :

R1 → R4 : Elles concernent le nom, la version, le domaine du problème et l'auteur du cas d'utilisation. Si le cas d'utilisation visuel est supporté par un outil, elles seront incorporées dans la propriété du diagramme et seront générées comme dans la documentation qui l'accompagne. En cas d'absence d'outil de modélisation, elles seront ajoutées manuellement sous forme de texte.

R5 → R6 : Chaque acteur sera modélisé par une partition (Swimlane) qui porte son nom.

R7 : Théoriquement dans la situation où un cas d'utilisation UC1 inclut un cas d'utilisation UC2, celle-ci devient son préalable et conditionne son exécution. Pour cette raison, nous avons décidé de transformer l'élément *Include* en un nœud d'activité portant le stéréotype *Include* et le placer juste après le nœud du début du cas d'utilisation visuel. De cette façon, il fait référence au cas d'utilisation inclus qui devrait s'exécuter avant de démarrer le cas d'utilisation qui l'inclut.

R8 : Quand le comportement d'un UC1 peut être étendu par le comportement d'un UC2, nous disons alors que l'UC2 étend l'UC1. Cette extension se déclenche à un moment précis, selon une condition prédéterminée lors de l'exécution des séquences d'étapes de l'UC1. Nous avons donc décidé de modéliser ce cas d'utilisation, qui étend le comportement d'un autre, par un nœud d'activité qui porte le stéréotype *Extend*.

R9 → R10 : la précondition et la postcondition seront transformées en une note attachée respectivement au nœud initial et au nœud final du cas d'utilisation visuel.

R11 → R13 : Les verbes d'action du patron de communication textuel seront transformés en des nœuds d'actions dans le cas d'utilisation visuel. Leur emplacement, dans les partitions (Swimlanes), variera en fonction du type du patron de communication.

R14 : *Data* ou donnée dans le patron de communication se transforme en une propriété incorporée dans le nœud de contrôle ou dans le nœud d'objet. Il est tout à fait possible d'incorporer Data dans le nœud d'action juste à droite du verbe d'action.

R15 → R16 : Les deux éléments *Object* et *Destination* du patron de communication textuel seront transformés en des nœuds d'objets dans le cas d'utilisation visuel.

R17 : Le mot clé *And* se transforme en un nœud de jointure dans le cas d'utilisation visuel.

R18 → R19 : Ces deux règles ont pour vocation de générer un nœud de décision dans le cas d'utilisation visuel. Elles se génèrent à chaque fois qu'un scénario alternatif se déclenche dans le cas d'utilisation textuel (R18) ou dans le cas où il est possible d'appliquer un verbe d'action sur deux objets, un qui appartient au scénario nominal et l'autre appartient au scénario alternatif (R19).

R20 : Cette règle montre que l'instance d'un objet est associée à l'instance d'un autre objet. Cette règle se manifeste au moment de l'utilisation du verbe d'action *Associate*. Dans le cas d'utilisation visuel, cette règle s'illustre par deux nœuds d'objets successifs dont le premier prend le statut de *Associated*.

R21 : Cette règle montre que l'instance d'un objet est dissociée de l'instance d'un autre objet. Cette règle se manifeste au moment de l'utilisation du verbe d'action *Dissociate*. Dans

le cas d'utilisation visuel, cette règle s'illustre par deux nœuds d'objets successifs dont le premier prend le statut de *Dissociated*.

Tableau 8.11 Règles de transformation du cas d'utilisation textuel en cas d'utilisation visuel

#	Cas d'utilisation textuel	Cas d'utilisation visuel
R1	Use case name	Nom du cas d'utilisation visuel
R2	Version	Doivent être mentionnés dans la propriété du cas d'utilisation si le cas d'utilisation est supporté par un outil de modélisation, si non sous forme de texte qui accompagne le diagramme visuel.
R3	Problem domain	
R4	Author	
R5	Primary Actor	*Swimlane* de l'acteur primaire.
R6	Secondary Actor	*Swimlane* de l'acteur secondaire
R7	Include	Une activité stéréotypée <<Include>> qui référence le cas d'utilisation inclut doit être placée juste après le nœud de début.
R8	Extend	Une activité stéréotypée <<Extend>> en référence au cas d'utilisation qui étend le comportement du cas d'utilisation étendu. Elle doit être placée au point d'extension approprié.
R9	Precondition	Une note attachée au nœud initial ou une propriété du nœud initial dans l'outil de modélisation.
R10	Postcondition	Une note attachée au nœud final ou une propriété du nœud final dans l'outil de modélisation.
R11	Verbe d'action de l'acteur primaire	Action dans le *Swimlane* de l'acteur primaire
R12	Verbe d'action de l'acteur secondaire	Action dans le *Swimlane* de l'acteur secondaire
R13	Verbe d'action du système	Action dans le Swimlane du système
R14	Data	Incorporé dans le *Object Flow* ou le *Control Flow*. Il est possible de l'incorporer dans le nœud d'action juste après le verbe d'action.
R15	Object	Nœud d'objet
R16	Destination	Nœud d'objet
R17	And	Noeud de bifurcation (Fork node)
R18	Or	Noeud de décision (Decision Node)
R19	Decision	Noeud de décision (Decision Node)

R20	Verbe d'action *Associate*	Deux nœuds d'objets successifs. Le premier nœud d'objet prend le statut *Associated*. Ce statut montre que l'instance du premier objet est associée/assignée à l'instance du deuxième objet.
R21	Verbe d'action Dissociate	Deux nœuds d'objets successifs. Le premier nœud d'objet prend le statut *Dissociated*. Ce statut montre que l'instance du premier objet est dissociée de l'instance du deuxième objet.

8.8 Synthèse

Ce chapitre présente deux manières de documenter un cas d'utilisation, une textuelle et l'autre visuelle, ainsi que la méthode permettant de transformer un cas d'utilisation textuel en un cas d'utilisation visuel.

La documentation d'un cas d'utilisation textuel s'appuie sur un nouveau gabarit composé de treize éléments. Ce gabarit est muni de deux patrons de communication, le premier sert à exprimer l'interaction de l'acteur principal et de l'acteur secondaire et l'autre, à exprimer l'interaction du système. Ces deux patrons de communication sont dotés d'une proposition de quelques verbes génériques pouvant exprimer toutes les actions de l'acteur et du système dans un cas d'utilisation. De plus, des règles de restriction permettant de vérifier, lors de la rédaction des cas d'utilisation, les opérations non autorisées et d'en informer l'analyste d'affaires de la raison de restriction ont été proposées.

La documentation d'un cas d'utilisation visuel s'appuie sur un nouveau profil UML2, basé sur le diagramme d'activité UML2 et adapté à cette fin. Ce profil est accompagné des règles de contrôle permettant de respecter sa syntaxe.

Dans le but de garder un lien et une traçabilité entre les deux manières de documenter un cas d'utilisation, 21 règles de transformation ont été proposées afin de transformer un cas d'utilisation textuel en un cas d'utilisation visuel.

CHAPITRE 9

DU CAS D'UTILISATION VISUEL AU DIAGRAMME DE CLASSE D'ANALYSE

Ce chapitre présente la phase 5 de ce projet de recherche, soit la transformation du CIM en PIM en utilisant la technique de transformation des cas d'utilisation visuels vers le diagramme de classe d'analyse.

9.1 Procédure de transformation du cas d'utilisation visuel en Diagramme de classe d'analyse

La procédure de transformation d'un cas d'utilisation visuel au diagramme de classe se fait en trois étapes :

1. Une étape automatique qui consiste à appliquer des règles de transformation.
2. Une étape manuelle qui consiste à répondre à une série de questions pour découvrir de nouveaux éléments du diagramme de classe.
3. Une étape manuelle qui consiste à lier l'ensemble des portions générées du diagramme de classe.

9.1.1 Étape de transformation automatique

Cette étape a pour objectif de générer une portion du diagramme de classe à partir d'un cas d'utilisation visuel. Cette portion contient une classe-archétype *Moment-interval* et ses opérations, une classe-archétype *Party*, des classes non stéréotypées et des associations entre ces différentes classes.

Six règles de transformation réalisent cette étape automatique :

R1 : Pour chaque cas d'utilisation visuel, générer une classe archétype *Moment-Interval*. La classe *Moment-Interval* représente l'objet dynamique de ce cas d'utilisation, l'élément *Activity* de l'EBP et le composant *Business Process* du modèle de composant correspondant.

R2 : Le nom de la classe archétype *Moment-Interval* doit correspondre au verbe du cas d'utilisation.

R3 : Pour chaque *Swimlane* différent du *Swimlane* system faisant partie du système à automatiser, une classe-archétype *Party* qui porte le même nom du *Swimlane* doit être générée.

R4 : Pour chaque élément de type *Object* dans le cas d'utilisation visuel, une classe qui porte le même nom que cet objet est générée.

R5 : Toutes les actions du cas d'utilisation visuel seront transformées en des méthodes de la classe-archétype *Moment-Interval*.

R6 : Relier chaque classe générée par une association simple à la classe-archétype *Moment-Interval*.

9.1.2 Étape de transformation manuelle guidée par des questions

Cette étape consiste à enrichir et consolider le diagramme de classe, généré automatiquement à l'étape un, par d'autres classes qui seront découvertes en répondant pas à pas aux questions. Celles-ci guident la procédure de transformation manuelle de l'étape deux.

Règles procédurales

A – Découvrir *MI-Detail*

Pour chaque classe *Moment-Interval* Archétype :

A1 : a-t-elle besoin d'être supportée par une classe qui détaille ses items ?

Si oui

A2 : créer une classe-archétype *MI-Detail*. Ensuite, relier-la par une relation d'agrégation à la classe *Moment-Interval*.

A3 : mettre les cardinalités :

A3.1 : Cardinalité ''1'' du côté *Moment-Interval*

A3.2 : Cardinalité ''1..*'' du côté *Mi-Détail*

B – Découvrir les archétypes *Place* et *Thing*

B1 : existe-t-il un archétype *Place* ?

B2 : existe-t-il un archétype *Thing* ?

Si oui

B3 : stéréotyper chaque classe par l'archétype *Place* ou *Thing*.

C : Découvrir et spécifier les archétypes *Roles* des archétypes *Party*, *Place* et *Thing* (PPT)

Pour chaque archétype PPT :

C1 : participe-t-il par un rôle spécifique en interaction avec le *Moment-Interval* archétype ?

Si oui

C1.1 : Est-il relié à l'archétype *Moment-Interval* ou à *Mi-Detail* (s'il existe) ?

C1.2 : Relier l'archétype *Role* à *Moment-Interval* ou *MI-Detail* et *PPT* par une association simple

Mettre les cardinalités

Si

Liaison entre le *Role* et le *Moment-Interval* :

Alors

C1.3.1 : Cardinalité ''0..*'' du côté *Moment-Interval*

C1.3.2 : Cardinalité ''1'' du côté *Role*

Si non

Liaison entre le *Role* et PPT

C1.3.3 : Cardinalité ‘’0..1’’ du côté *Role*

C1.3.4 : Cardinalité ‘’1’’ du côté PPT

Si non

Liaison entre le *Role* et *MI-Detail*

C1.3.5 : Cardinalité ‘’0..*’’ du côté *Mi-Detail*

D – Découvrir l’archétype *Description*

D1 : existe-t-il une classe-archétype *Description* ?

D2 : Décrit-elle l’archétype *Moment-Interval*, *Role* ou les archétypes PPT ?

Si oui

D3 : Relier l’archétype *Description*, selon la réponse de la question D2, au PPT, *Role* ou à *Moment-Interval*, par une association simple.

Mettre les cardinalités

D3.1 : Cardinalité ‘’1’’ du côté de l’archétype *Description*

D3.2 : Cardinalité ‘’0..*’’ du côté des archétypes *Moment-Interval* et *Role* et ‘’0..1’’ du côté des archétypes PPT

E – Spécifier les attributs et les méthodes des archétypes

Pour chaque archétype :

E1 : sélectionner les attributs et les méthodes des éléments génériques de chaque archétype du DNC;

E2 : Spécifier et/ou adapter les noms génériques des attributs et des méthodes à la situation étudiée;

E3 : Ajouter des attributs et des méthodes spécifiques à la situation si nécessaire.

Fin de la procédure

9.1.3 Étape de fusion des portions de diagrammes de classe

Gérer la traçabilité entre les différents cas d'utilisation constituant le modèle des cas d'utilisation et leurs diagrammes de classe correspondants peut être une activité difficile. Dans ce cas-ci, selon l'approche préconisée dans notre recherche, il est plus raisonnable de travailler séparément sur chaque cas d'utilisation et y générer son diagramme de classe équivalent. En effet, chacun de ces diagrammes de classe représente une portion du diagramme de classe intégral. Cette stratégie, représente une solution pragmatique, surtout dans le cas où plusieurs équipes d'analystes travaillent en parallèle sur différents cas d'utilisation. Il est judicieux, dans ce cas-ci, d'appliquer une technique intégratrice qui facilite l'obtention du modèle de classe intégrale.

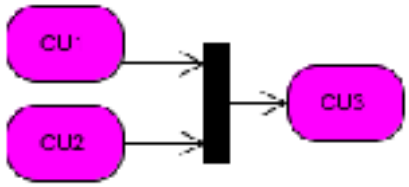
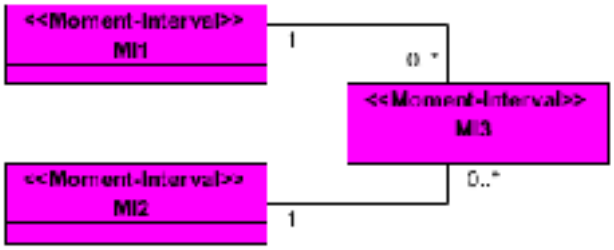
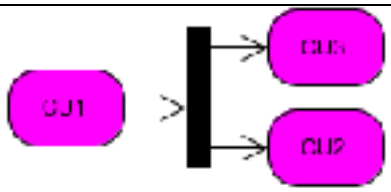
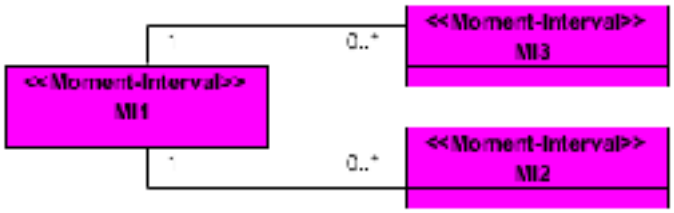
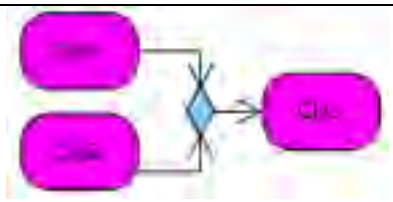
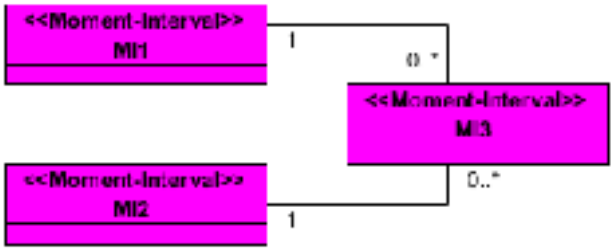
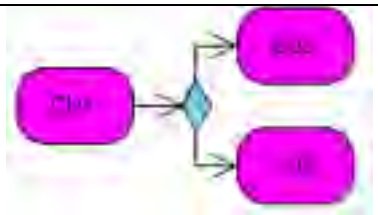
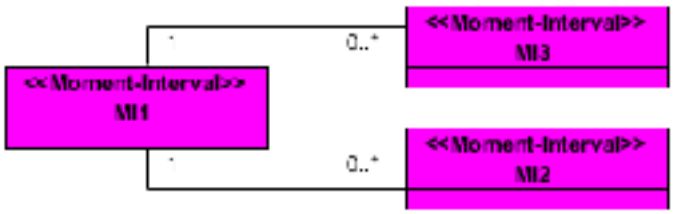
Nous présentons une technique qui consiste à intégrer les portions du diagramme de classe en un seul diagramme global. Celle-ci se base à la fois sur la succession des cas d'utilisation transactionnels et sur le principe de fusion de classe que nous présentons sous forme de deux règles R1 (Link MI) et R2 (Merge Classe).

R1 : Link MI

1. Identifier chaque classe-archétype *Moment-Interval* de chaque diagramme de classe généré;
2. Lier les classes-archétypes *Moment-Interval* entre elles, par une association simple, selon l'ordre de la succession des cas d'utilisation correspondants dans le modèle de cas d'utilisation. Concernant les cas d'utilisation liés par les nœuds de contrôles, le Tableau 9.1 résume les liens d'association entre les classes-archétypes *Moment-Interval*.
3. Mettre les cardinalités suivantes sur les associations entre les classes-archétypes *Moment-Interval* :
    - Cardinalité ‘’1’’ du côté *Prior-MI* (pour chaque classe qui précède respectivement l'autre);

- Cardinalité ‘’0..*’’ du côté *Next-MI* (pour chaque classe qui suit respectivement l’autre).

Tableau 9.1 Association entre Moment-Interval selon l’étape 3 de la règle R1 (LINK MI)

Liens entre cas d’utilisation	Association entre les classes Moment-Interval
	
	
	
	

R2 : Merge Class

Cette règle consiste à rechercher dans chaque portion générée du diagramme de classe les classes qui appartiennent à plus d’une portion, ensuite identifier les liens qui les associent

aux autres classes, et enfin les relier ensemble par des associations simples. De cette manière, le diagramme de classe global peut se structurer davantage via cette forme d'association.

En effet, la règle R2 se résume à :

1. Rechercher les classes similaires dans chaque portion du diagramme de classe.
2. Pour chaque deux portions du diagramme de classe A et B, supprimer une seule classe similaire C d'une des deux portions du diagramme de classe A ou B.
3. Relier la classe non supprimée avec les classes qui étaient directement liées avec la classe supprimée C.

CHAPITRE 10

CAS D'ÉTUDE ET VÉRIFICATION DE LA MÉTHODOLOGIE DE TRANSFORMATION DU CIM EN PIM

Ce chapitre présente la phase 6 de ce projet de recherche, soit l'expérimentation, à l'aide d'un cas d'étude, de l'ensemble de la méthodologie de transformation du CIM en PIM que nous avons proposées dans cette thèse.

10.1 Conception du modèle de processus d'affaires

Pour s'assurer de l'applicabilité de notre méthodologie de transformation du CIM en PIM, nous avons conduit une expérimentation en adoptant la méthode d'étude de cas. Pour ce faire, nous avons utilisé le cas de EU-RENT Car (ANNEXE I).

Selon notre analyse, nous recensons deux modèles de processus d'affaires que nous avons jugé importants pour notre étude. Le premier, consacré au service à la clientèle servant à louer une voiture et le second servant à entretenir une voiture.

10.1.1 Conception du modèle de processus d'affaires du service à la clientèle

Selon notre analyse des besoins du service à la clientèle d'EU-RENT, nous avons identifié 5 EBP, 2 acteurs (Actors), 5 objets ressources (Resources) et 1 objet résultat (Result).

En appliquant les deux étapes P1 et P2 de la procédure de conception du BPM (5.1.8) nous obtiendrons :

- l'acteur *Client* est responsable des EBP *Reserve*, *Rent*, *Return* et *Pay*;
- l'acteur *Clerk* est responsable des EBP *Assign* et *Inspect*;
- les objets ressources sont *Branch*, *Car*, *Credit Card* et *Insurance*;
- l'objet résultat est *Inspection Report*.

Ensuite, avec l'application des trois dernières étapes P3, P4 et P5 de la procédure de conception du BPM (5.1.8), nous obtiendrons le modèle illustré à la Figure 10.1.

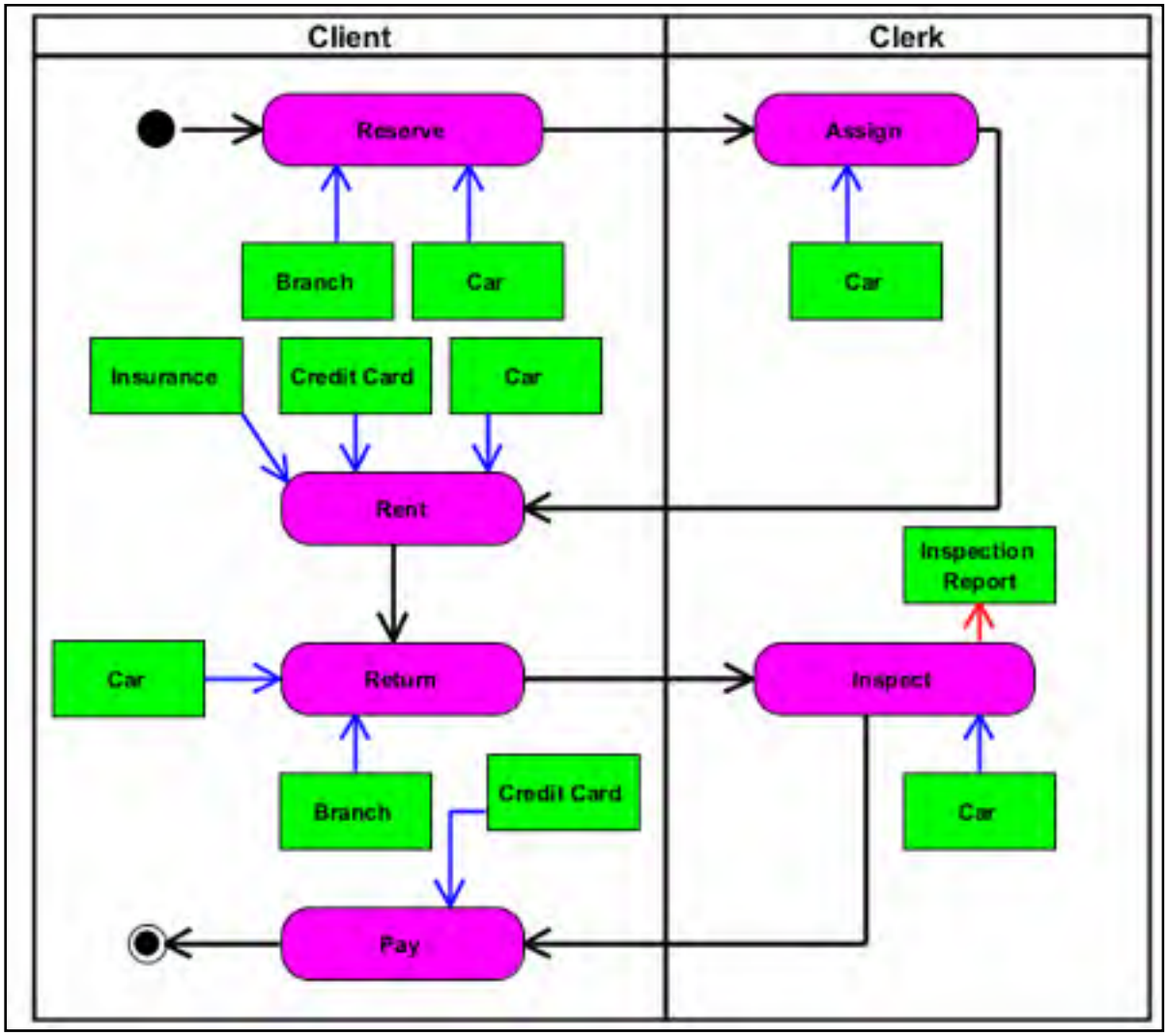


Figure 10.1 Modèle de processus d'affaires de service à la clientèle

10.1.2 Conception du modèle de processus d'affaires du service maintenance

Selon notre analyse des besoins du service maintenance EU-RENT, nous avons identifié 5 EBP, 4 acteurs (Actors), 2 objets ressources (Resources) et 3 objets résultats (Results).

En appliquant les deux étapes P1 et P2 de la procédure de conception du BPM (5.1.8) nous obtiendrons :

- l'acteur *Mce Department Head* est responsable de l'EBP *Maintenance Schedule*;
- l'acteur *Mce Lead Hand* est responsable des EBPs *Prepare Work Order* et *Make Available*;
- l'acteur *Garage Mechanic* est responsable de l'EBP *Maintain Car*;
- l'acteur *Manager* est responsable de l'EBP *Sell*;
- les objets ressources sont *Branch*, *Car* et *Garage Mechanic Planning*;
- les objets résultats sont *Service Request*, *Work Order* et *Maintenance Report*.

Ensuite, avec l'application des trois dernières étapes P3, P4 et P5 de la procédure de conception du BPM (5.1.8), nous obtiendrons le modèle illustré à la Figure 10.2.

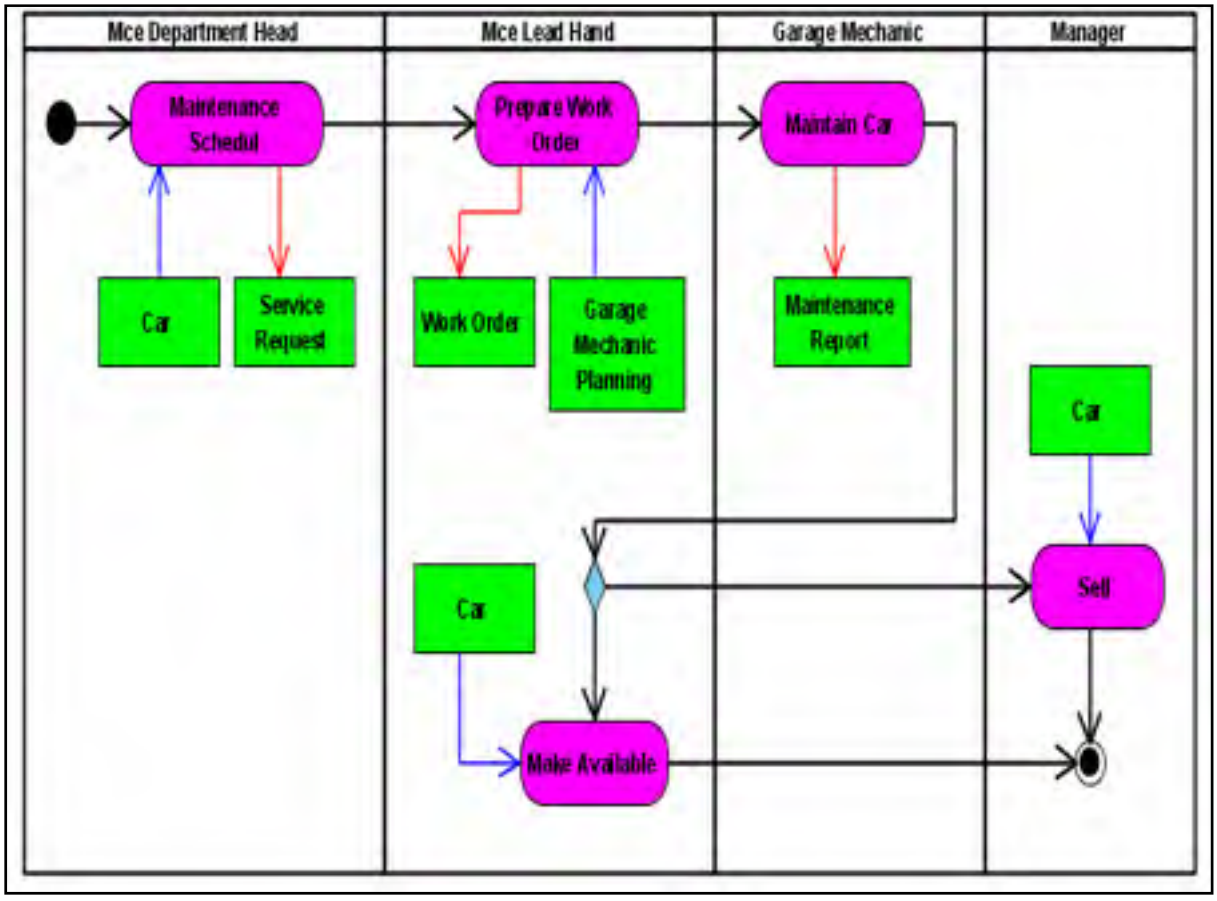


Figure 10.2 Modèle de processus d'affaires de service maintetance

10.2 Transformation du BPM en BCM

10.2.1 Transformation du BPM « Service à la Clientèle » au BCM

10.2.1.1 Application des règles de transformations automatiques

En appliquant les neuf règles de transformations automatiques (de RA1 jusqu'à RA9 dans la chapitre 6.6), nous obtiendrons le modèle des composants d'affaires illustré à la Figure 10.3.

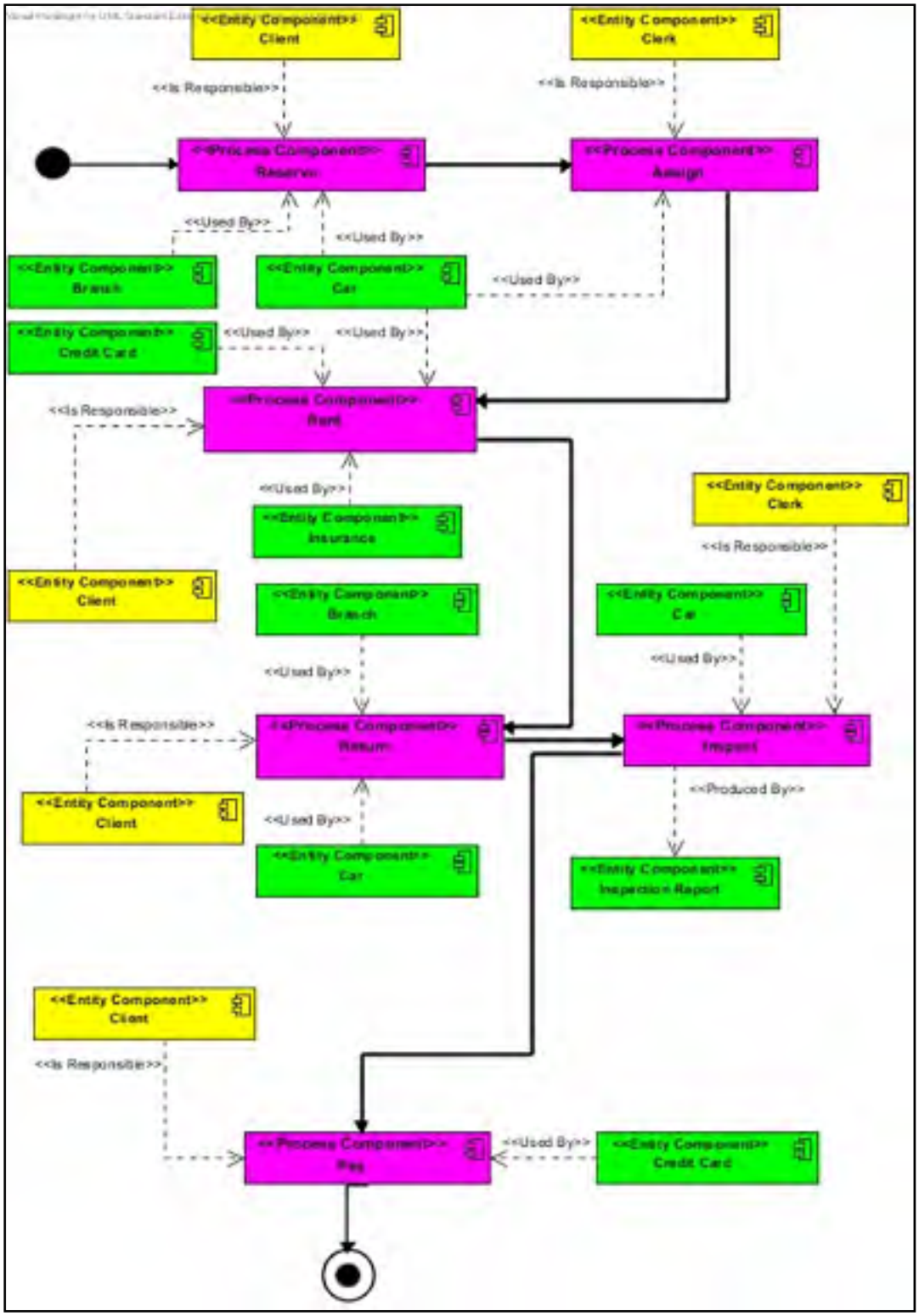


Figure 10.3 Modèle des composants d'affaires Service à la Clientèle EU-RENT Car

10.2.1.2 Application de la règle de transformation manuelle

Pour appliquer la règle de transformation manuelle RA10 (chapitre 6.6), il faut passer en revue les différents composants d'affaires constituant ce modèle. Cet exercice requiert une analyse qui permet de détecter le ou les composants qui représentent un niveau de granularité jugé faible par rapport aux autres composants. Cette activité est subjective et ne peut être réalisée que par des analystes expérimentés.

Dans cette étude de cas, notre analyse nous conduit à constater que le composant *Credit Card* est de granularité relativement faible en comparaison avec les autres composants du modèle de composant d'affaires *Réservation* EU-RENT Car. Généralement une carte de crédit est la propriété physique d'une personne, elle ne peut donc pas exister sans cette personne car elle en fait partie intégrante. En suivant la règle RA10, nous avons décidé de l'incorporer dans le composant d'affaires *Client*. Donc, un *Package* sera créé et incorporera les deux *Entity Components Client* et *Credit Card* selon la règle RA10. La Figure 10.4 montre le nouveau modèle de composant d'affaires après application de la règle manuelle RA10.

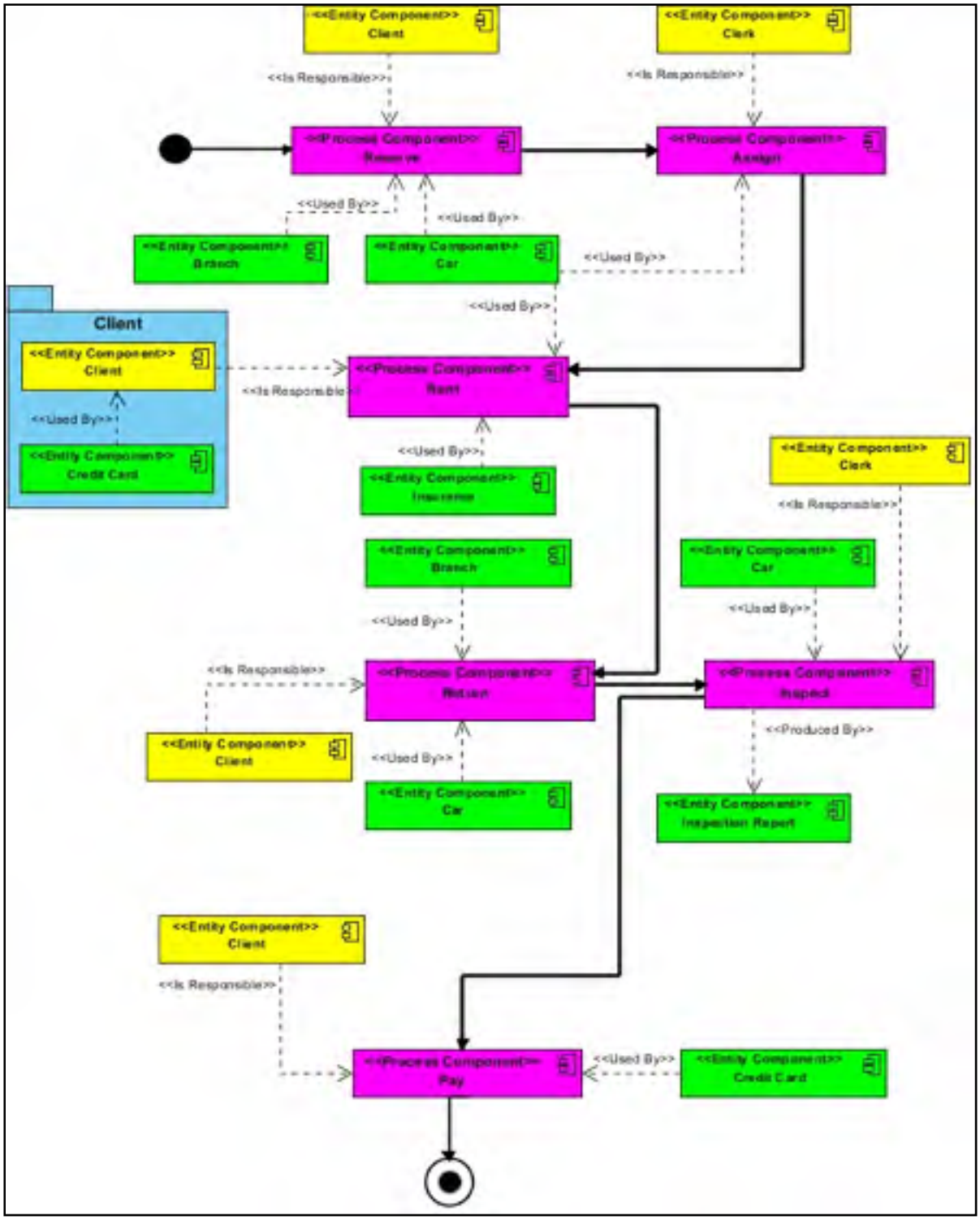


Figure 10.4 Modèle de composant d'affaires Service à la Clientèle EU-RENT Car après application de la règle RA10

10.2.2 Transformation du BPM du service maintenance ‘’Maintain Car’’ au BCM

En appliquant les neuf règles de transformations automatiques (de RA1 jusqu'à RA9 – chapitre 6.6), nous obtenons le modèle des composants d’affaires illustré à la Figure 10.5.

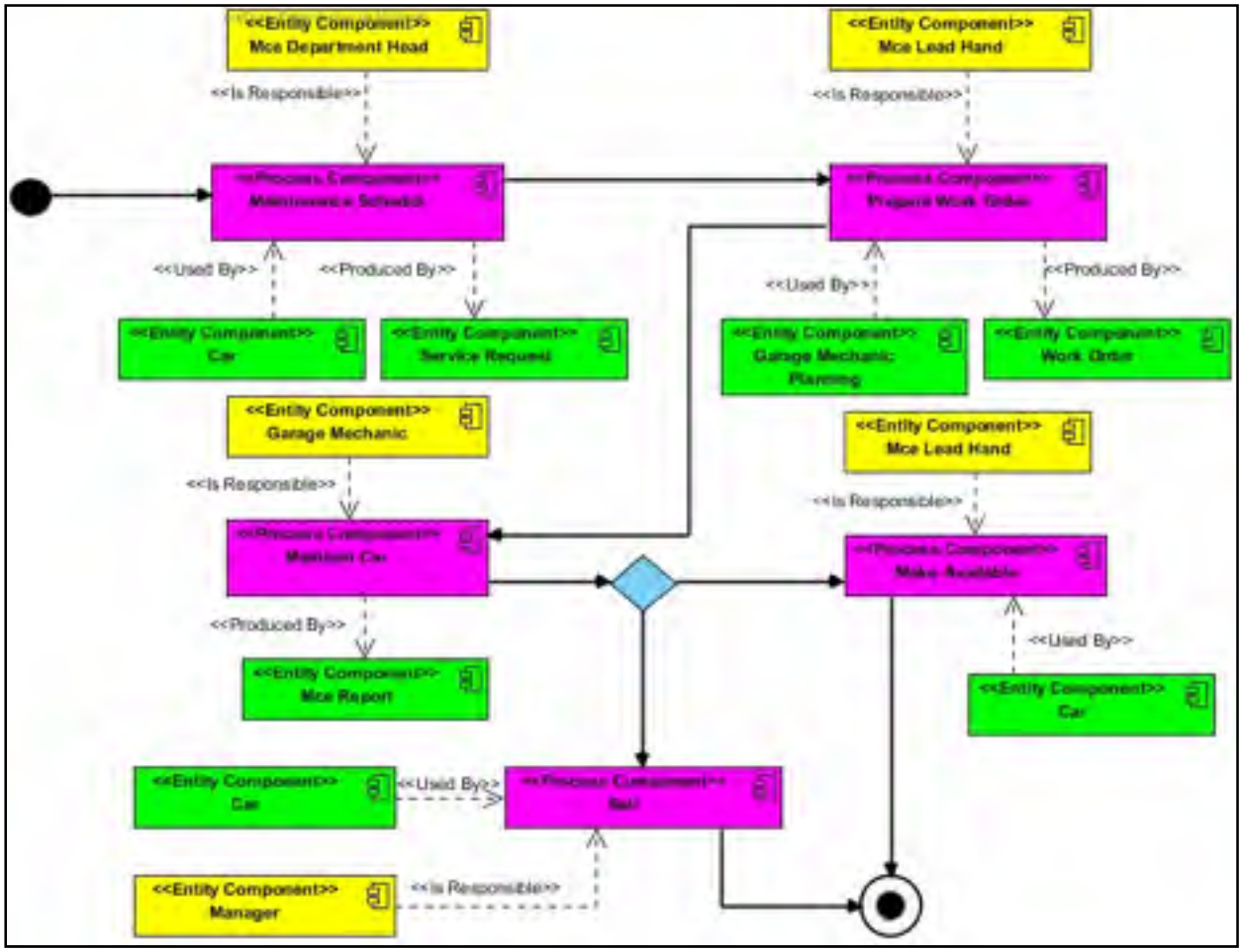


Figure 10.5 Modèle de composants d’affaires du service maintenance

Après avoir analysé le BCM généré comme illustré à la Figure 10.5, nous n’avons constaté aucun composant d’affaires dont la granularité est faible par rapport aux autres composants d’affaires constituant le modèle BCM *Maintain Car*.

Il n’est donc pas nécessaire d’appliquer la règle RA10 dans ce cas-ci.

10.2.3 Discussion

Le but principal du modèle des composants d'affaires est d'aligner davantage la modélisation des processus d'affaires aux autres activités de spécifications logicielles. Le BCM est supposé contribuer aux méthodes et techniques déjà existantes dans ce domaine.

En outre, et mise à part la valeur ajoutée du BCM dans le cycle de développement logiciel, il faut préciser que le BCM offre les avantages suivants :

- Il est basé sur des patrons réutilisables, configurables et facilement maintenables;
- Il s'aligne aux modèles de processus d'affaires et représente un intrant pour construire le modèle de composant système;
- Il tire profit des avantages de l'orienté-objet;
- Il utilise le standard de modélisation UML2 via un nouveau profil BCM;
- Il représente une contribution supplémentaire à l'activité de spécification logicielle basée sur la modélisation des processus d'affaires;
- Il représente une référence pour modéliser le diagramme de classe d'analyse.

10.3 Du modèle des processus d'affaires au modèle des cas d'utilisation

Le but de cet exercice d'expérimentation est de transformer les deux BPM, Service à la clientèle et Service de maintenance d'une voiture de l'EU-Rent Car, en un seul modèle de cas d'utilisation. Les deux BPM sont illustrés respectivement à la Figure 10.1 et la Figure 10.2.

En effet, selon notre proposition, il est impératif d'identifier les *Activities* de chaque BPM qui seront candidats à se transformer en cas d'utilisation.

10.3.1 BPM Service à la clientèle

Nous avons constaté pendant notre activité de catégorisation des *Activities* du BPM Service à la clientèle qu'il en existe deux, supportée et automatique :

- Les *Activities* supportés sont *Reserve*, *Rent*, *Return*, *Inspect* et *Pay*;
- L'*Activity Assign* est automatique car c'est le système logiciel de service à la clientèle qui assigne les voitures disponibles aux différentes réservations. Il s'agit d'une action temporisée qui, à un moment précis de la journée, déclenche l'activité de l'assignation d'une voiture à une réservation.

Nous avons identifié deux acteurs dans le BPM soit *Client* et *Clerk*.

En résumé, nous avons six *Activities* du BPM qui seront transformées en cas d'utilisation dans l'UCM et deux acteurs du BPM en acteurs de l'UCM.

10.3.2 BPM service de maintenance (Maintain Car)

En appliquant la même analyse effectuée sur le BPM service à la clientèle, nous constatons que toutes les *Activites* du BPM Maintenance d'une voiture à savoir *Maintenance Schedule*, *Prapare Work Order*, *Maintain Car*, *Make Available* et *Sell* sont candidates à se transformer en cas d'utilisation. Elles sont toutes de type *Activity* supportées. Les acteurs *Mce Department Head*, *Mce Lead Hand*, *Garage Mechanic* et *Manager* sont candidats à se transformer en acteurs de l'UCM.

10.3.3 Application des règles de transformation (chapitre 7.1)

10.3.3.1 Application des règles de transformation R1 et R2

En appliquant les deux règles de transformation R1 et R2 nous obtenons le résultat suivant : toutes les *Activities* identifiées lors de notre analyse des deux BPM seront transformées en des cas d'utilisation de l'UCM.

Voici les cas d'utilisation générés à partir des deux règles de transformation R1 et R2 (Tableau 10.1) :

Tableau 10.1 Résultat de l'application des règles de transformation R1 et R2

Activity (supportée et automatique) du BPM	Cas d'utilisation de l'UCM
Rent	Rent
Return	Return
Inspect	Inspect
Pay	Pay
Assign	Assign
Maintenance Schedul	Maintenance Schedul
Prepare Work Order	Prepare Work Order
Maintain Car	Maintain Car
Make Available	Make Available
Sell	Sell

10.3.3.2 Application de la règle de transformation R3

Le Tableau 10.2 résume les acteurs transformés du BPM en UCM. Chaque acteur sera associé à un cas d'utilisation avec lequel il interagira.

Tableau 10.2 Résultat de l'application de la règle de transformation R3

Acteurs de BPM	Acteur de l'UCM	
		Cas d'utilisation associés
Client	Client	Reserve, Rent, Return, Pay
Clerk	Clerk	Assign, Inspect
Mce Department Head	Mce Department Head	Maintenance Schedul
Mce Lead Hand	Mce Lead Hand	Prepare Work Order, Make Available
Garage Mechanic	Garage Mechanic	Maintain Car
Manager	Manager	Sell

10.3.3.3 Application de la règle de transformation R4

Les objets identifiés dans les deux BPM sont *Car*, *Branch*, *Credit Card*, *Insurance*, *Inspection Report*, *Service Request*, *Work Order*, *Garage Mechanic*, *Garage Mechanic Planning* et *Maintenance Report*. Ils seront tous liés aux cas d'utilisation générés à partir de l'application des deux règles de transformation R1 et R2.

10.3.3.4 Application de la règle de transformation R5

L'application de la règle R5 se résume à placer les nœuds de contrôle dans l'UCM selon leur ordre d'apparition dans le BPM.

10.3.4 La construction du diagramme de cas d'utilisation

La conception du diagramme d'activités s'appuie sur le métamodèle proposé à la (7.2.2) ainsi que sur les éléments générés à partir des règles de transformation R1, R2, R3 et R4.

Il faut mentionner qu'après une analyse préliminaire des différents cas d'utilisation générés, nous avons constaté l'apparition de trois acteurs secondaires *Bank*, *Mce Lead Hand* et *Mce Department Hand*. Ceux-ci interagiront respectivement avec les cas d'utilisation *Pay*, *Prepare Work Order* et *Maintain Car*.

Il est à noter que les deux acteurs *Mce Lead Hand* et *Mce Department Hand* sont à la fois des acteurs principaux et secondaires dans l'UCM. L'UCM est illustré à la Figure 10.6.

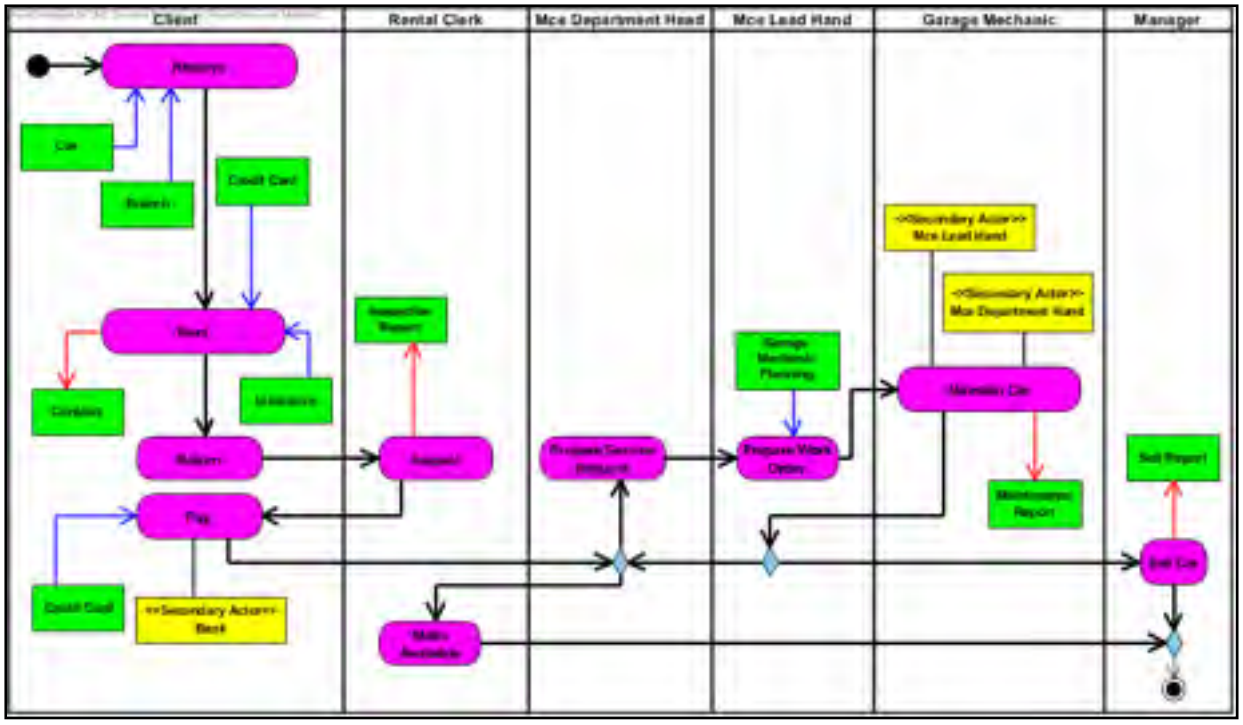


Figure 10.6 UCM de l'EU-RENT Car

10.3.5 Discussion

Il est clair que notre UCM présente une grande différence par rapport à l'UCM traditionnel qui se réalise à l'aide du diagramme de cas d'utilisation UML. En effet, la différence majeure entre les deux approches, l'approche traditionnelle et la nôtre, se situe au niveau de l'introduction des nouveaux éléments et de l'adoption du diagramme d'activités comme formalisme de modélisation.

Nous sommes conscients que cette approche changera partiellement les habitudes des analystes d'affaires. À contrario, nous sommes convaincus qu'elle permet de représenter plus d'éléments dans l'UCM tels les objets étant déjà identifiés dans le BPM et la façon d'enchainer les cas d'utilisation. De plus, notre approche permet de garder une traçabilité entre les éléments du BPM et de l'UCM.

Il est en effet possible d'automatiser les règles de transformation R1, R2, R3, R4 et R5 à l'aide d'un des langages de transformation de modèles QVT ou ATL.

10.4 Du cas d'utilisation textuel au cas d'utilisation visuel

Dans cette section, nous discuterons l'applicabilité de notre approche par l'expérimentation de notre démarche d'écriture des cas d'utilisation textuels et de leurs transformations en des cas d'utilisation visuels.

10.4.1 Procédure d'expérimentation

D'après le modèle de cas d'utilisation EU-Rent Car illustré à la Figure 10.6, nous recensons dix cas d'utilisation, six acteurs primaires et trois acteurs secondaires. Ces cas d'utilisation, de type transactionnels, constituent notre cadre d'expérience.

La procédure de notre expérimentation suit ces étapes :

- Détailler chaque cas d'utilisation à l'aide du gabarit d'écriture de cas d'utilisation (Tableau 8.1);
- Vérifier la cohérence du gabarit de cas d'utilisation par les règles de restriction du Tableau 8.8;
- Appliquer les règles de transformation du cas d'utilisation textuel en cas d'utilisation visuel (Tableau 8.11);
- Vérifier la cohérence du cas d'utilisation visuel à l'aide des règles de contrôle (Tableau 8.10);
- Dresser un bilan qui résume cette transformation.

Les dix cas d'utilisation sont expérimentés à la section suivante.

10.4.1.1 Cas d'utilisation Reserve

Cas d'utilisation textuel Reserve

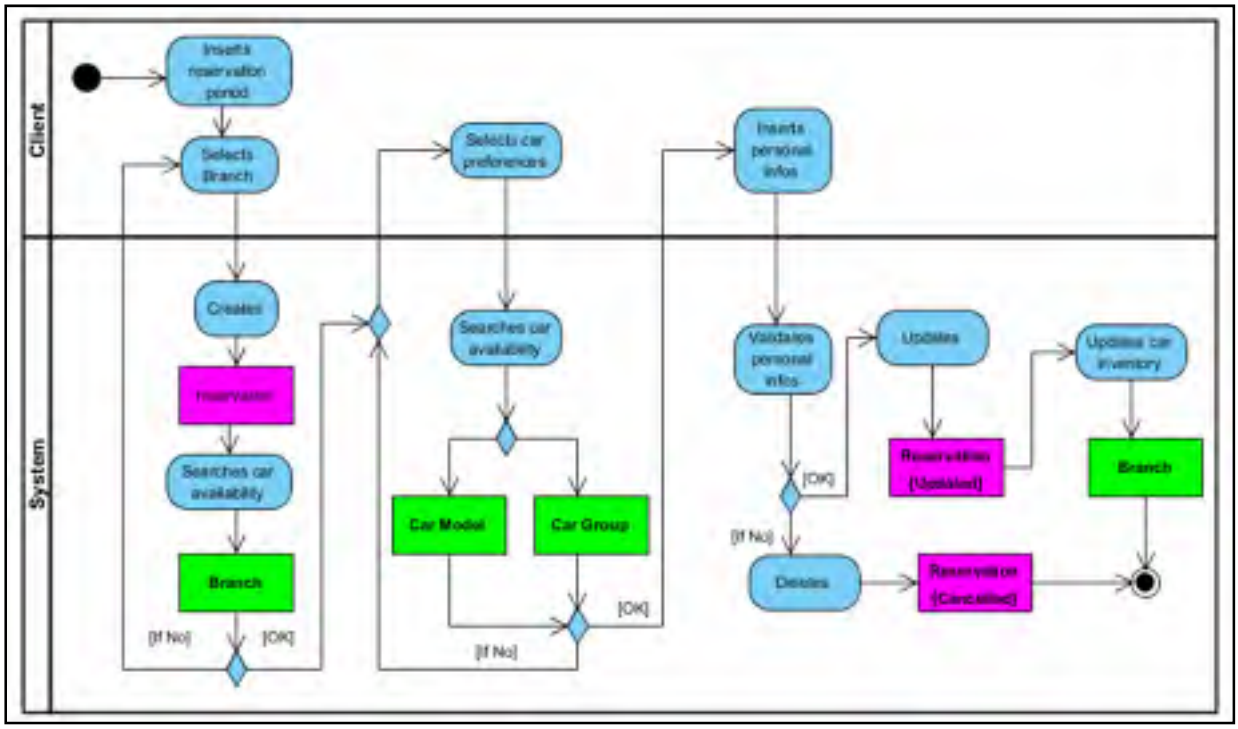
Use case Name : Reserve On-line	**Version** : 1.0
Problem Domain : EU Rental – advance reservation	**Author** : Samir Kherraf
Purpose and Description: Clients need to make a reservation to ensure the availability of the car category and car model that they want, at the pick-up branch that they want.	
Actors	**Primary actor**: Client **Role of primary actor**: Make a reservation. In the context of a phone reservation system, the client would be dealing with a Clerk who simply acts as an intermediary between the client and the system.
Dependency	Car Model, Car Group, Branch.
Trigger events	The Client's desire to make a reservation.
Preconditions	None
Postconditions	Client has a reservation for a particular car group, a particular duration, with a particular pickup branch, and a particular drop-off branch. The inventory of available cars of a particular car group at a particular branch for a particular branch is diminished by one for the duration of the rental The inventory of available cars of that car group at the drop-off branch is increased by one on the projected date of drop-off

Main Scenario

Step	Client	System	Decision
1	Inserts reservation period		
2	Selects Branch		
3		Creates Reservation	
4		Searches car availability into a Branch	If No
5	Selects car preferences		
6		Searches car availability into Car Models Or Car Group	If No
7	Inserts personal infos		
8		Validates personal infos	If No
9		Updates Reservation	

10		Updates car inventory into Branch	
Alternate Scenarios			
Alternate Step	**Alternate client action**	**Alternate system action**	**Alternate decision**
2	Inserts reservation period		
4	Selects car preferences		
6		Deletes Reservation	

Cas d'utilisation visuel Reserve



10.4.1.2 Cas d'utilisation Rent

Cas d'utilisation textuel Rent

Use case Name: Rent	**Version** : 1.0
Problem Domain: EU Rental	**Author** : Samir Kherraf
Purpose and Description: Clients finalize the rental contract when they pick up a car that they had reserved.	

Actors	**Primary** : Client **Role of primary actor**: Make a rental.
Dependency	Contract, Credit Card, Insurance, Car.
Trigger events	The Client's showing up at branch on day of rental
Preconditions	The client has a reservation on the given date at the given branch
Postconditions	Contract is created and signed. Car is out of lot.

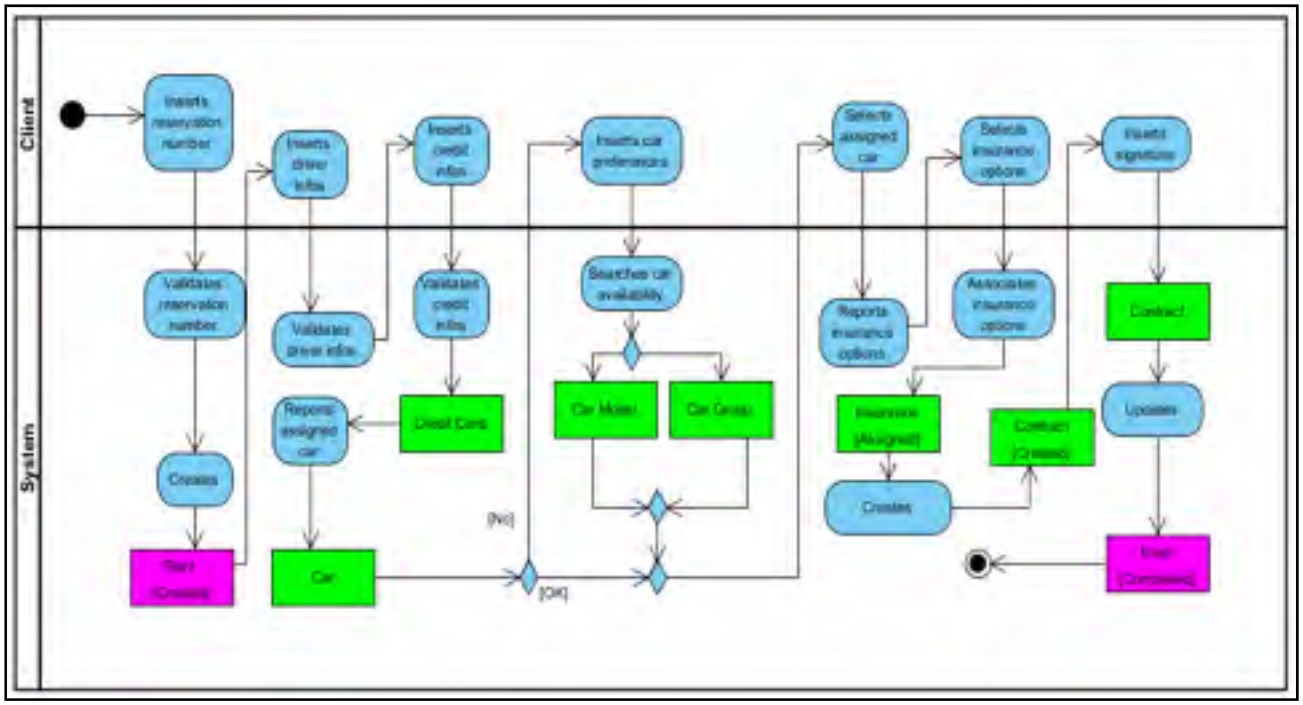
Main Scenario

Step	Client	System	Decision
1	Inserts reservation number		
2		Validates reservation number	
3		Creates Rent	
4	Inserts driver information		
5		Validates driver information	
6	Inserts credit information		
7		Validates credit information into Credit Card	
8		Reports assigned car from Car	If No Go to 8
9	Selects assigned car		
		Reports insurance options	
10	Selects insurance options		
11		Associates insurance options Into Insurance	
12		Creates Contract	
13	Inserts signature into Contract		
14		Updates Rent	

Alternate Scenarios

Alternate Step	Alternate client action	Alternate system action	Alternate decision
8	Inserts car preferences		
		Searches car availability Into Car Group Or Car Model	If Ok Go to Step 9

Cas d'utilisation visual Rent



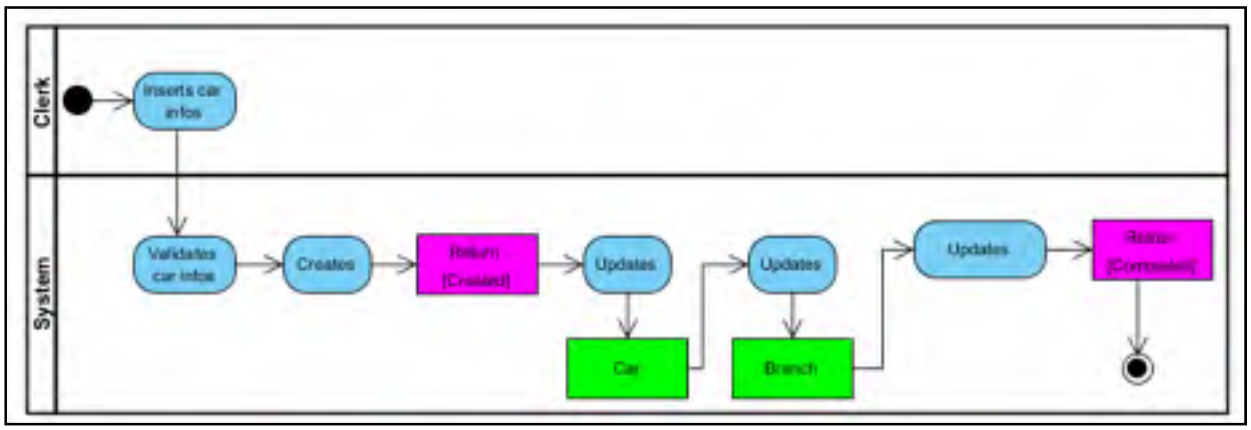
10.4.1.3 Cas d'utilisation Return

Cas d'utilisation textuel Return

Use case Name : Return		**Version** : 1.0	
Problem Domain : EU Rental		**Author** : Samir Kherraf	
Purpose and Description : Clients return car into branch			
Actors	**Primary** : Clerk **Role of primary actor**: Return car rental.		
Dependency	None		
Trigger events	The Client's return car into branch.		
Preconditions	The client has a rental car.		
Postconditions	The car is ready to inspect		
Main Scenario			
Step	**Clerk**	**System**	**Decision**
1	Inserts car information		

2		Validates car information	
3		Creates Return	
4		Updates Car	
5		Updates Branch	
6		Updates Return	

Cas d'utilisation visuel Return



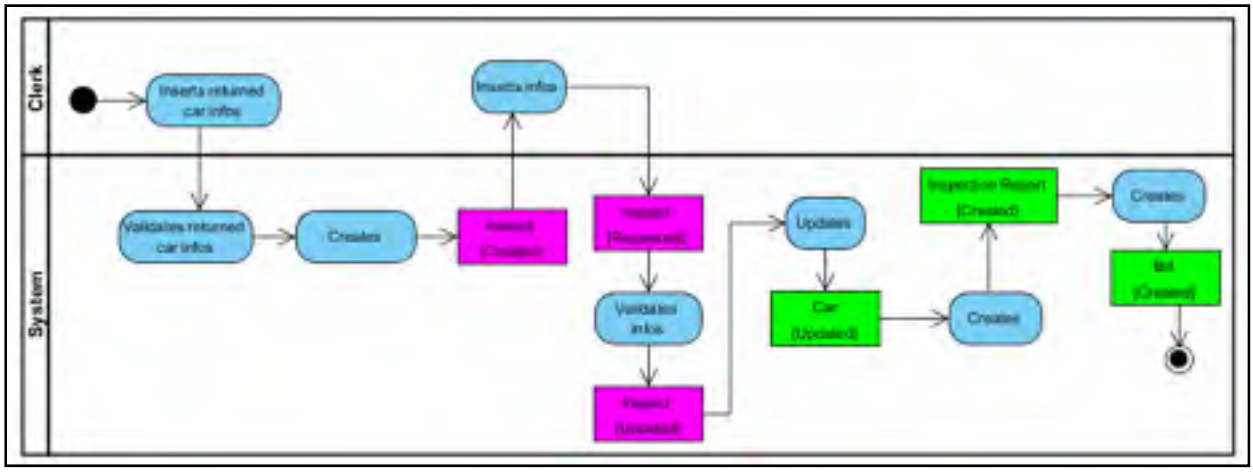
10.4.1.4 Cas d'utilisation Inspect

Cas d'utilisation textuel Inspect

Use case Name : Inspect			**Version** : 1.0
Problem Domain : EU Rental			**Author** : Samir Kherraf
Purpose and Description: Clients return car into branch and Clerk inspects car			
Actors	**Primary** : Clerk **Role of primary actor**: Inspect car rental.		
Dependency	Inspection Report, Bill		
Trigger events	The Client's return car into branch.		
Preconditions	The client has a rental car.		
Postconditions	Inspection report and bill are generated.		
Main Scenario			
Step	**Client**	**System**	**Decision**

1	Inserts returned car infos		
2		Validates returned car infos	
3		Creates Inspect	
4	Inserts information into Inspection		
5		Validates information into Inspection	
6		Updates Car	
7		Creates inspection report	
8		Creates Bill	

Cas d'utilisation visuel Inspect



10.4.1.5 Cas d'utilisation Pay

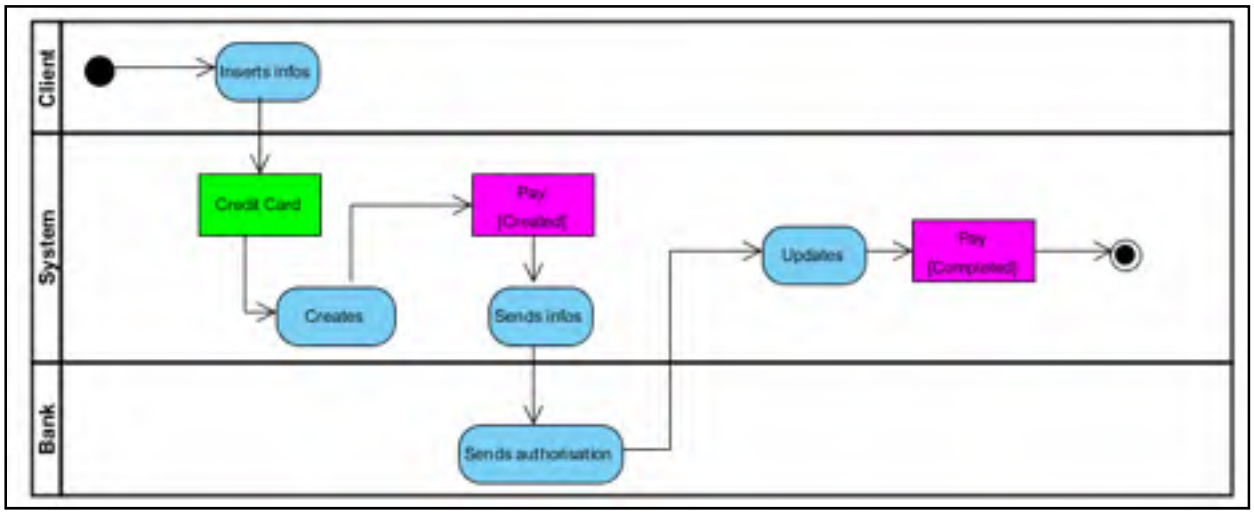
Cas d'utilisation textuel Pay

Use case Name : Pay	**Version** : 1.0
Problem Domain : EU Rental	**Author** : Samir kherraf
Purpose and Description: Clerks finalize the car inspection and produce inspection report.	
Actors	**Primary** : Client **Role of primary actor**: Paid car rental. **Secondary** : Bank **Role of secondary actor** : Verify fund and approve payment

Dependency	Credit Card,
Trigger events	The Client's desires to pay the amount of bill.
Preconditions	The client has a valid credit card, sufficient fund and bill rental
Postconditions	Client has a receive

Main Scenario				
Step	**Client**	**System**	**Bank**	**Decision**
1	Inserts information into credit card			
2		Creates Pay		
3		Sends information into Bank		
4			Sends authorisation	
5		Update Payment		

Cas d'utilisation visuel Pay

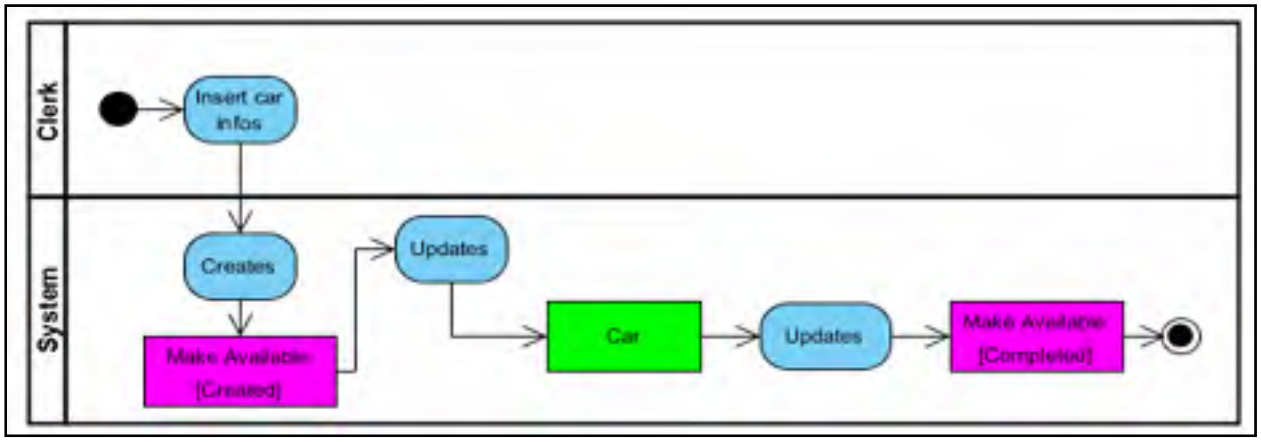


10.4.1.6 Cas d'utilisation Make Available

Cas d'utilisation textuel Make Available

Use case Name : Make Available		**Version** : 1.0	
Problem Domain : EU Rental		**Author** : Samir Kherraf	
Purpose and Description: Clerk make available car for rent.			
Actors	**Primary** : Clerk **Role of primary actor**: Make available car rental.		
Dependency	Inspect use case, Maintain Car use case.		
Trigger events	The bill is paid and the car is inspected or maintained.		
Preconditions	The car is inspected and doesn't need maintenance or maintained and is ready to rent.		
Postconditions	The car is available for rent.		
Main Scenario			
Step	**Clerk**	**System**	**Decision**
1	Inserts car information		
2		Creates Make Available	
3		Update Car	
4		Updates Make Available	

Cas d'utilisation visuel Make Available



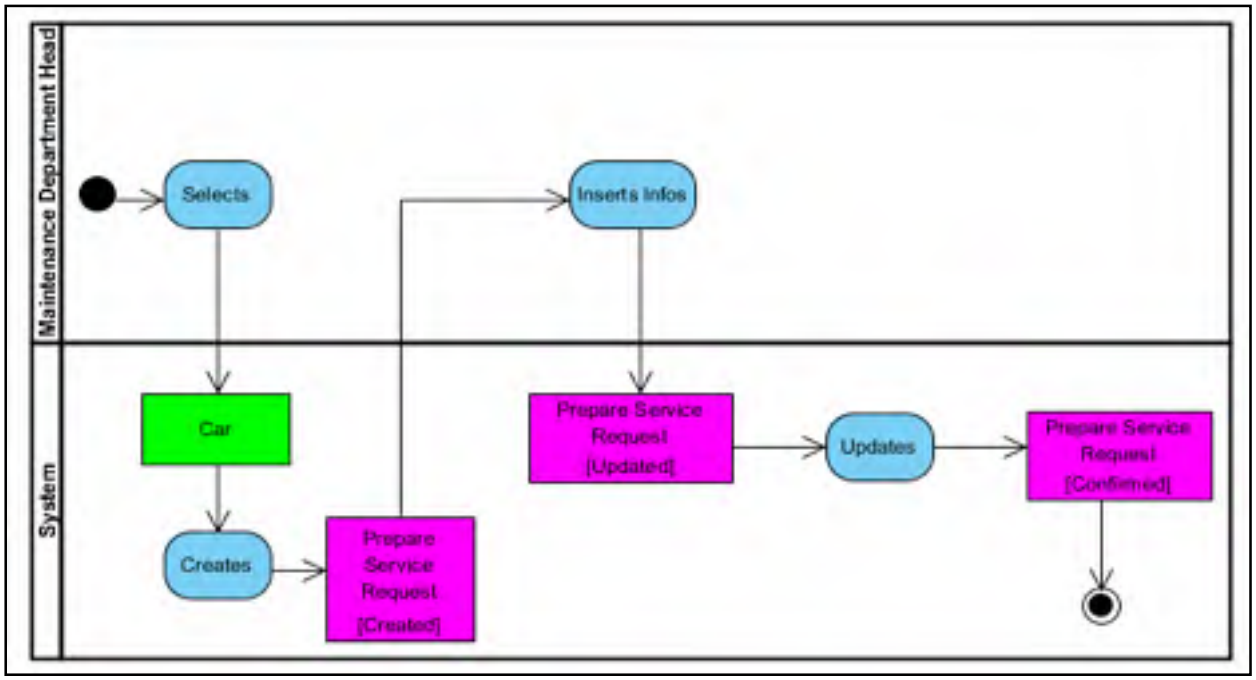
10.4.1.7 Cas d'utilisation Prepare Service Request

Cas d'utilisation Textuel Prepare Service Request

Use case Name : Prepare Service Request		**Version** : 1.0
Problem Domain : EU Rental – Prepare Service Request for maintain a car		**Author** : Samir Kherraf
Purpose and Description: Based on Inspection Report, the maintenance department head prepares a service request for maintain the car.		
Actors	**Primary actor**: Maintenance Department Head **Role of primary actor**: Prepares a service request. Responsible for maintenance of the fleet of cars.	
Dependency	Inspection use case	
Trigger events	After inspection, the car needs maintenance; Preventive maintenance.	
Preconditions	The car is not available for rent.	
Postconditions	The service request is rooted to Lead Hand Maintenance.	

Main Scenario			
Step	**Client**	**System**	**Decision**
1	Selects Car		
2		Creates Service Request	
3		Inserts Infos into Service Request	
5		Updates Service Request	

Cas d'utilisation visuel Prepare Service Request



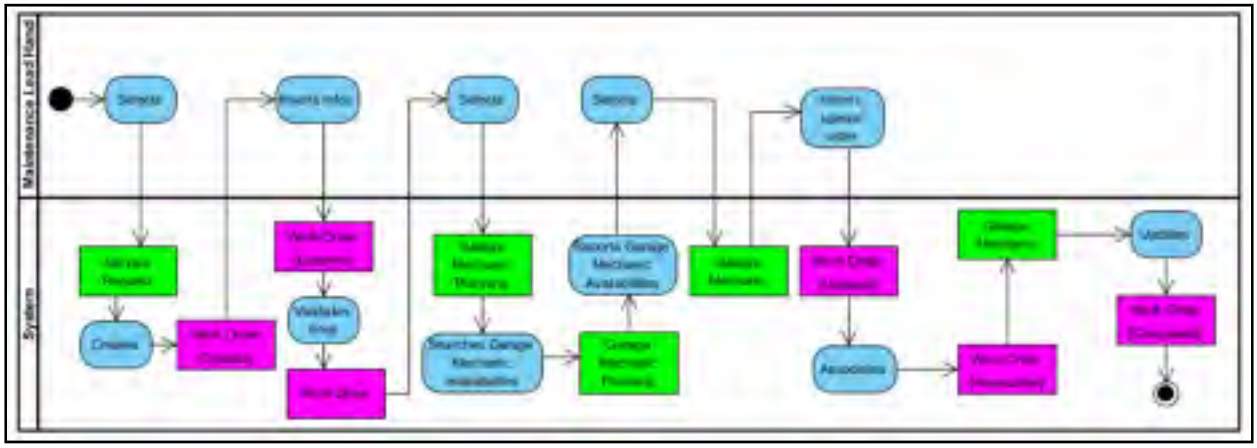
10.4.1.8 Cas d'utilisation Prepare Work Order

Cas d'utilisation textuel Prepare Work Order

Use case Name : Prepare Work Order			**Version** : 1.0
Problem Domain : EU Rental –Work Order for maintain a car			**Author** : Samir Kherraf
Purpose and Description: Based on Service Request, the maintenance Lead Hand prepares a Work Order and assign it a garage mechanic in order to maintain the car.			
Actors	**Primary actor**: Maintenance Lead Hand **Role of primary actor**: Prepares a Work Order. Manage a garages mechanics works.		
Dependency	Garage Mechanic Planning		
Trigger events	The Maintenance Lead Hand receives a Service Request.		
Preconditions	Service Request is received		
Postconditions	The Work Order is assigned to a Garage Mechanic.		
Main Scenario			
Step	**Client**	**System**	**Decision**
1	Selects Service Request		

2		Creates Work Order	
3	Inserts Infos into Work Order		
4		Validates infos into Work Order	
5	Selects Garage Mechanic planning		
6		Searches Garage Mechanic availabilities into Garage Mechanic Planning	
7		Reports Garage Mechanic Availabilities	
8	Selects Garage Mechanic		
9	Inserts special order into Work Order		
10		Associates Work Order into Garage Mechanic	
11		Updates Work Order	

Cas d'utilisation visuel Prepare Work Order

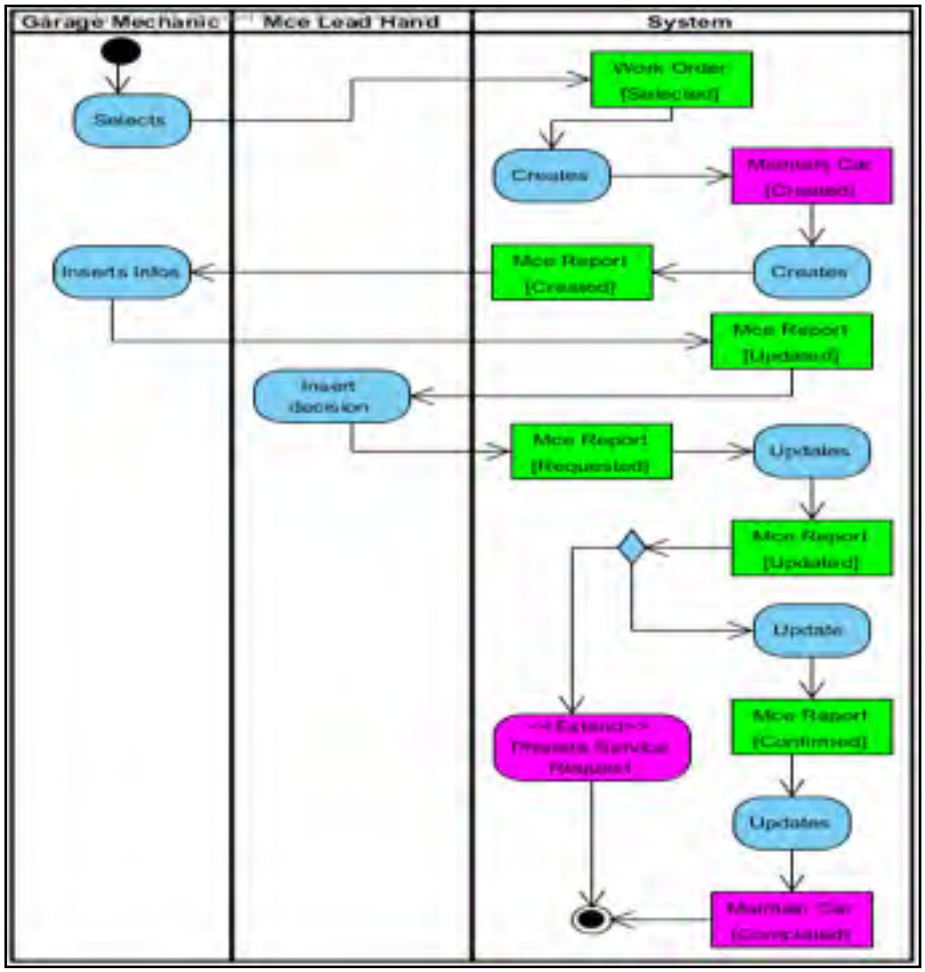


10.4.1.9 Cas d'utilisation Maintain Car

Cas d'utilisation textuel Maintain Car

Use case Name : Maintain Car				**Version** : 1.0
Problem Domain : EU Rental – Maintain Car				**Author** : Samir Kherraf
Purpose and Description: Based on Work Order, the Garage Mechanic executes the necessary tasks for maintain the car.				
Actors	**Primary actor**: Garage Mechanic **Role of primary actor**: Executes the Work Order **Secondary actor**: Maintenance Lead Hand **Role of secondary actor**: Verify executed works and validates the terminated Work Order			
Dependency	Maintenance Report			
Trigger events	The Work Order is received by the Garage Mechanic.			
Preconditions	The car is in the Garage. The Work Order is received by the Garage Mechanic according to his planning.			
Postconditions	The car is maintained or need more maintenance. The Work Order is completed. The service Request is completed.			
Main Scenario				
Step	**Garage Mechanic**	**Maintenance Lead Hand**	**System**	**Decision**
1	Selects Work Order			
2			Creates Maintain Car	
3			Creates Mce Report	
4	Inserts Infos Into Mce Report			
5		Inserts decision Into Mce Report		
6			Updates Mce Report	If No
7			Updates Mce Report	
8			Updates Maintain Car	
Alternate Scenarios				
Step	**Garage Mechanic**	**Maintenance Lead Hand**	**System**	**decision**
6	Extend Prepare Service Request Use Case			

Cas d'utilisation visuel Maintain Car



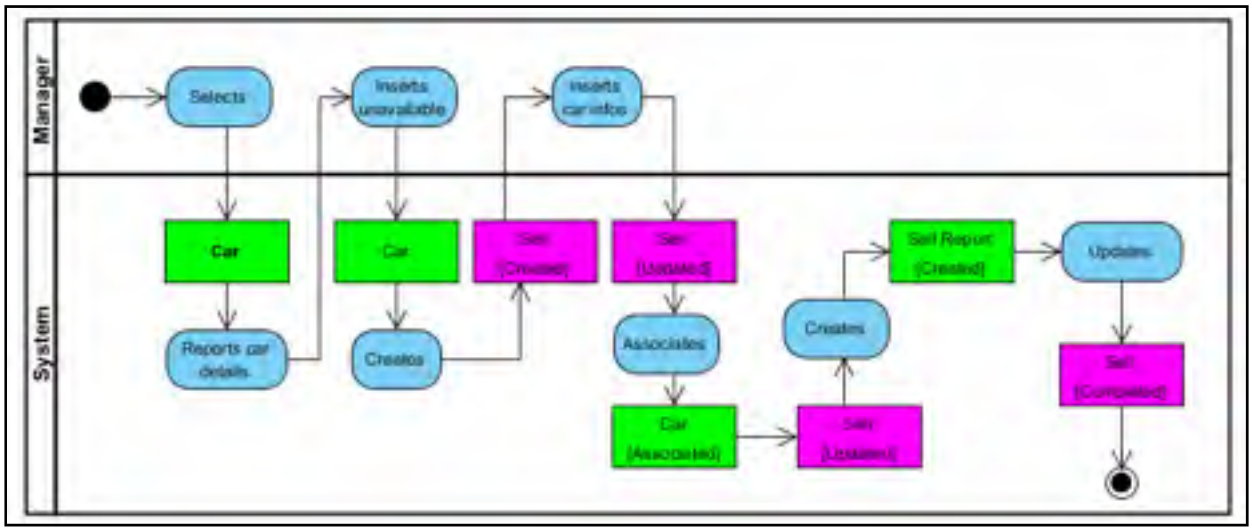
10.4.1.10 Cas d'utilisation Sell

Cas d'utilisation textuel Sell

Use case Name : Sell	**Version** : 1.0
Problem Domain : EU Rental	**Author** : Samir Kherraf
Purpose and Description: Manager registers the car as being "sold".	
Actors	**Primary** : Manager **Role of primary actor**: Sell a car.
Dependency	Sell Report
Trigger events	Car maintained and can not serve as a car rental.
Preconditions	The car is known and owned by EU-Rent.

Postconditions	The car is unavailable for rentals.		
Main Scenario			
Step	**Manager**	**System**	**Decision**
1	Selects Car		
2		Reports car details	
3	Inserts unavailable Iinto Car		
4		Creates Sell	
5	Inserts car info Into Sell		
6		Associates Car into Sell	
7		Creates Sell Report	
8		Updates Sell	

Cas d'utilisation visuel Sell



10.4.2 Évaluation et discussion des résultats

Sur les 19 règles de transformation que nous avons proposées, 13 d'entre elles seront évaluées. En fait, nous considérons que l'utilité des six autres règles de transformation se réduit à un mécanisme de transfert de la documentation d'un cas d'utilisation textuel vers un cas d'utilisation visuel. Cette documentation, qui est sous format textuel dans le cas

d'utilisation textuel, se transforme via ces règles de transformation en des notes attachées aux éléments du cas d'utilisation visuel. Dans la situation où ce cas d'utilisation visuel est supporté par un outil de modélisation, il sera plus facile d'incorporer cette documentation dans la propriété de chaque élément constituant ce cas d'utilisation visuel.

Les règles de transformations suivantes ne seront donc pas prises en compte dans notre évaluation :

- **R1** : le nom de cas d'utilisation reste le même dans les deux types de cas d'utilisation;
- **R2, R3 et R4** : qui sont respectivement la version du cas d'utilisation, le nom de son auteur et la description de son domaine de problème restent inchangés dans les deux cas d'utilisation textuel et visuel;
- **R9 et R10** : qui sont la précondition et la postcondition seront transformés en des notes attachées respectivement au nœud initial et au nœud final du cas d'utilisation visuel. Nous avons décidé de ne pas les représenter dans nos expériences afin de ne pas surcharger les diagrammes visuels.

10.4.2.1 Applicabilité des règles de transformation

Notre analyse d'applicabilité des règles de transformation se réduit en 13 règles : R5, R6, R7, R8, R11, R12, R13, R14, R15, R16, R17, R18 et R19. Elles seront évaluées dans le Tableau 10.3.

En fait, il se peut qu'un cas d'utilisation ne contienne pas tous les éléments préconisés dans le gabarit d'écriture des cas d'utilisation. L'utilisation de ces éléments dépend, comme illustré dans nos expériences, de la nature de chaque cas d'utilisation. Par exemple, il y a un cas qui nécessite une étape de décision et d'autres cas n'en ont pas besoin. De ce fait, Il est important de mentionner que l'application de chacune des règles de transformation est conditionnelle à la complétude d'un cas d'utilisation textuel (un cas d'utilisation qui contient tous les éléments définis dans son métamodèle).

Nous avons établi trois critères qui permettent de mesurer cette applicabilité :

1. **A (Applicable)** : un élément candidat à la transformation existe dans le gabarit d'écriture de cas d'utilisation. Ainsi la règle de transformation s'applique sur cet élément et génère l'élément correspondant souhaité, sans aucun problème, dans le cas d'utilisation visuel.
2. **NA (Non applicable)** : l'élément candidat à la transformation n'existe pas dans le gabarit d'écriture de cas d'utilisation. Donc, la règle de transformation ne s'applique pas. Il est à noter que la non-applicabilité d'une règle n'influence aucunement sur la démarche de transformation.
3. **E (Erreur)** : un élément candidat à la transformation existe dans le gabarit d'écriture de cas d'utilisation. Ainsi la règle de transformation s'applique sur cet élément et génère avec quelques problèmes l'élément correspondant souhaité dans le cas d'utilisation visuel, ou ne réussit pas à le générer.

Tableau 10.3 Applicabilité des règles de transformation permettant de transformer des cas d'utilisation textuels vers des cas d'utilisation visuels

		Cas d'utilisation									
		Reserve	Rent	Return	Inspect	Pay	Make Available	Prepare Service Request	Prepare Work Order	Maintain Car	Sell
Règles de transformations	**R5**	A	A	A	A	A	A	A	A	A	A
	R6	NA	NA	NA	NA	A	NA	NA	NA	A	NA
	R7	NA	NA	NA	NA	NA	NA	NA	NA	NA	NA
	R8	NA	NA	NA	NA	NA	NA	NA	NA	A	NA
	R11	A	A	A	A	A	A	A	A	A	A
	R12	NA	NA	NA	NA	NA	NA	NA	NA	A	NA
	R13	A	A	A	A	A	A	A	A	A	A
	R14	A	A	A	A	A	A	A	A	A	A
	R15	A	A	A	A	A	A	A	A	A	A
	R16	A	A	NA	A	A	NA	A	A	A	A
	R17	NA	NA	NA	NA	NA	NA	NA	NA	NA	NA
	R18	A	NA	NA	NA	NA	NA	NA	NA	NA	NA
	R19	A	A	NA	NA	NA	NA	NA	NA	A	NA

10.4.2.2 Discussion des résultats

D'après les résultats du Tableau 10.3, nous remarquons que les règles de transformation R5, R11, R13, R14, R15 s'appliquent à 100% des cas d'utilisation. Cela est dû à l'existence des éléments, considérés fondamentaux, dans le cas d'utilisation textuel. En effet, il est primordial qu'un cas d'utilisation incorpore les éléments suivants : acteur primaire, verbe d'action de l'acteur primaire, verbe d'action de l'acteur système, données et objets.

La règle de transformation R16 s'applique dans huit cas d'utilisation sur dix. Les deux cas d'utilisation *Return* et *Make Available* n'ont pas d'objet *Destination*, ce qui explique la non applicabilité de cette règle.

Les règles de transformation R6, R8, R12, R18 sont appliquées une seule fois et R19 trois fois. En effet, cela peut s'expliquer par l'existence non systématique des éléments candidats à la transformation du cas d'utilisation textuel.

Dans notre expérience nous n'avons pas eu recours aux éléments *Include* et *And*, ce qui explique la non applicabilité des deux règles de transformation, R7 et R17, dans l'ensemble des cas d'utilisation. La non utilisation de l'élément *Include* se justifie par notre approche de conception du diagramme de cas d'utilisation : ce diagramme est basé sur le BPM, constitué d'un ensemble d'activités qui s'enchaînent, où une activité ne peut démarrer que lorsque l'activité précédente est terminée. De ce point de vue, chaque activité inclut implicitement le comportement de l'activité précédente.

10.5 Du cas d'utilisation visuel au diagramme de classe d'analyse

10.5.1 Expérimentation de l'étape de transformation automatique

En appliquant les six règles de transformation automatiques (chapitre 9.1.1) sur les différents cas d'utilisation visuels, nous obtiendrons les diagrammes de classe des Figure 10.7 à 10.26.

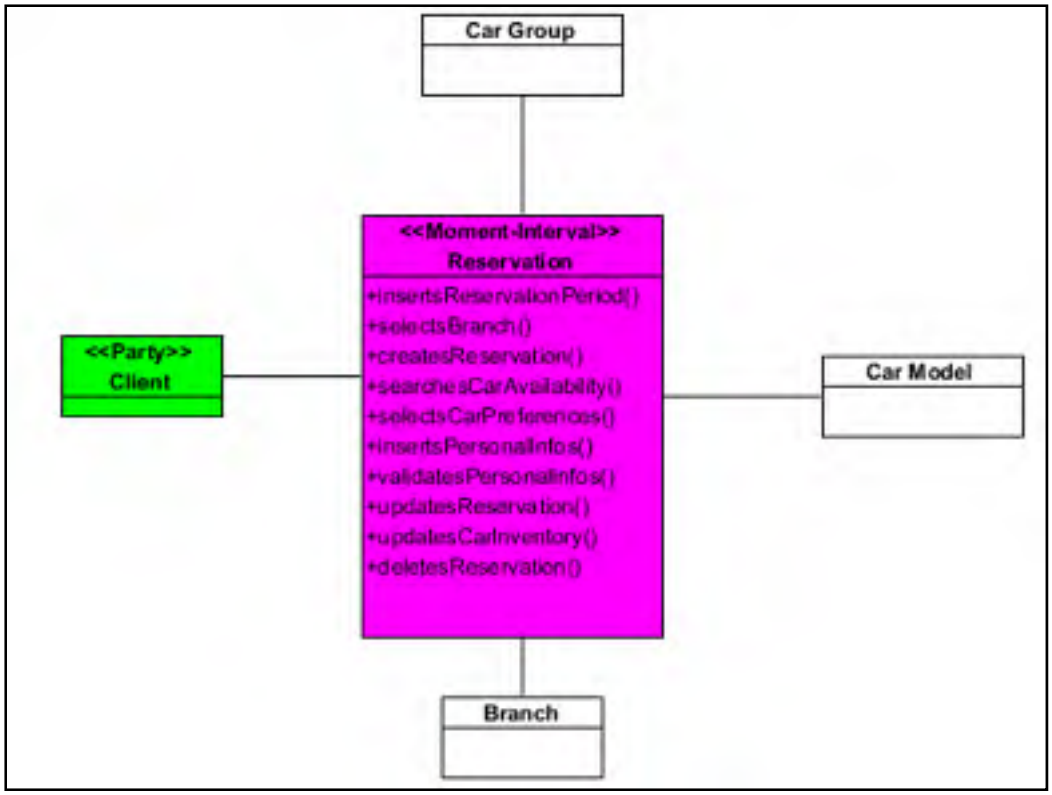


Figure 10.7 Diagramme de classe Reserve

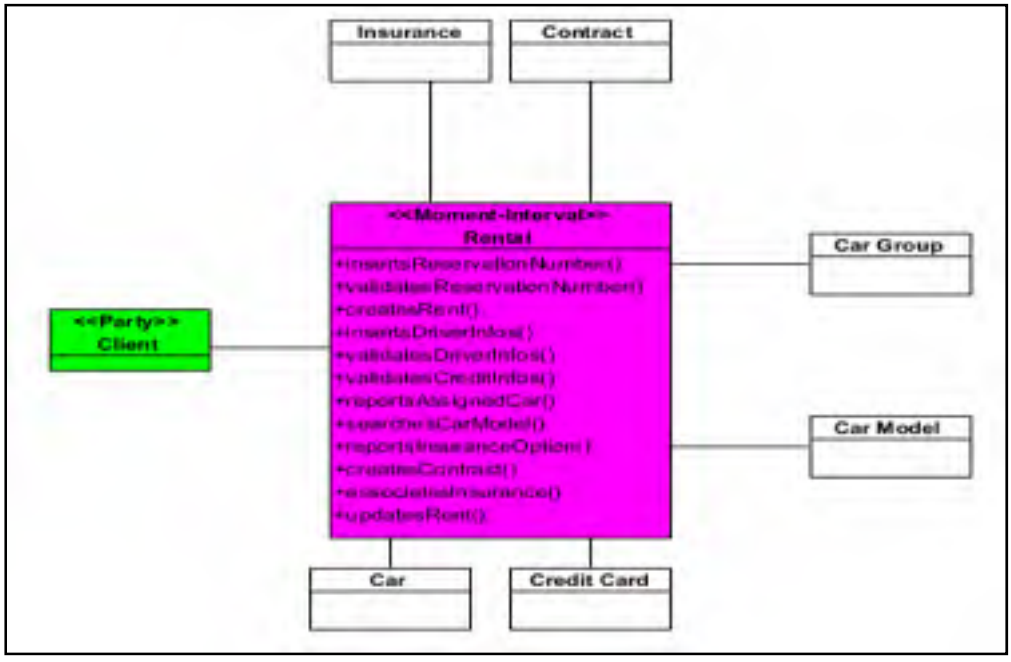


Figure 10.8 Diagramme de classe Rent

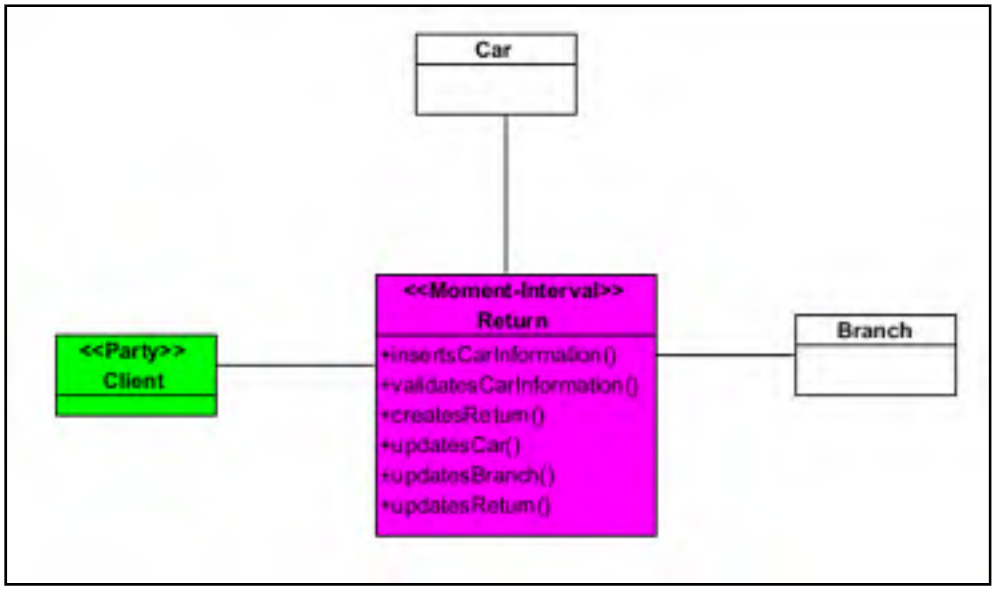


Figure 10.9 Diagramme de classe Return

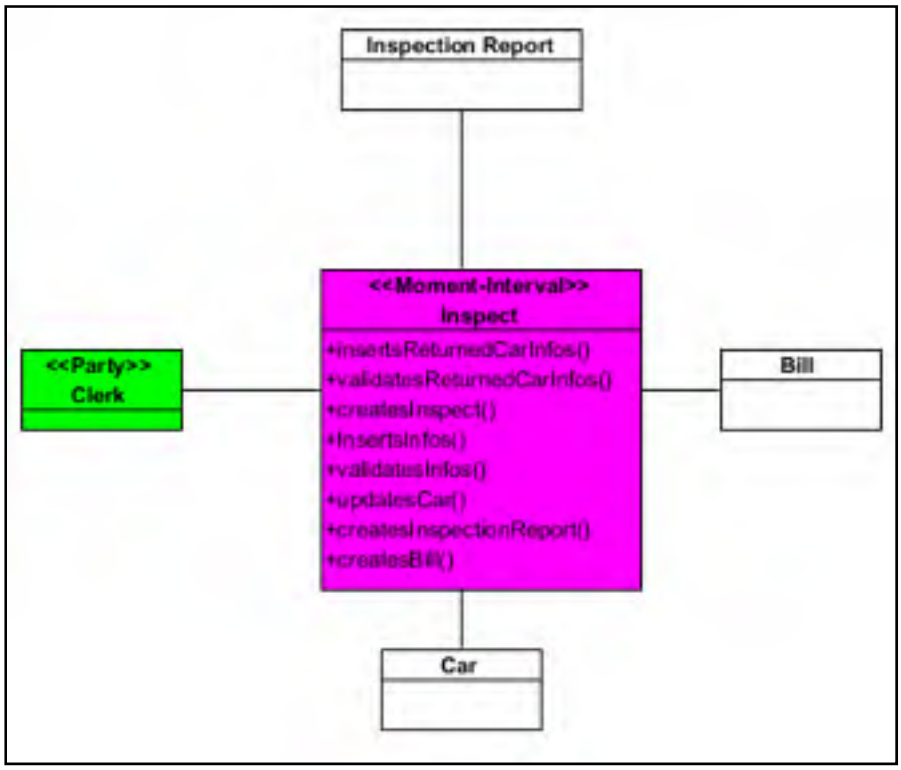


Figure 10.10 Diagramme de classe Inspect

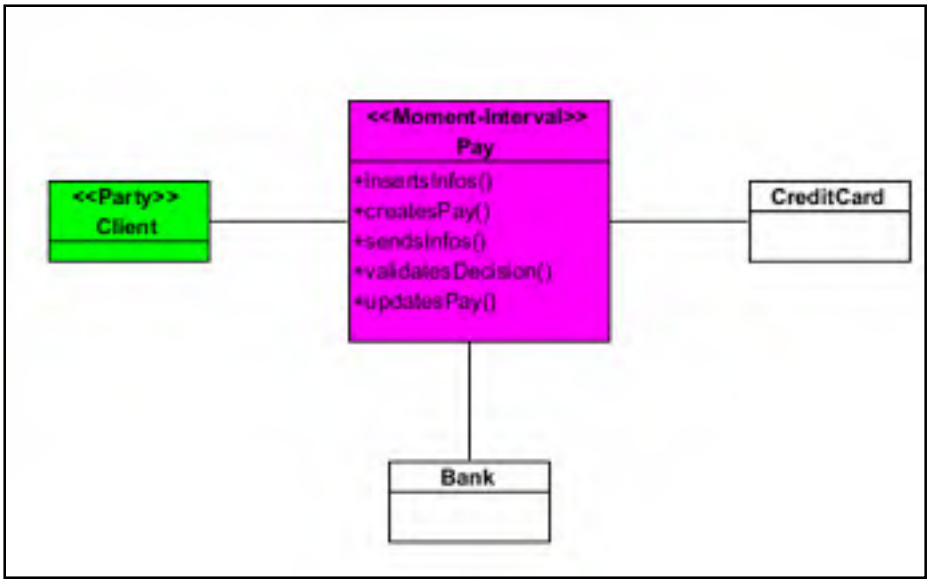


Figure 10.11 Diagramme de classe Pay

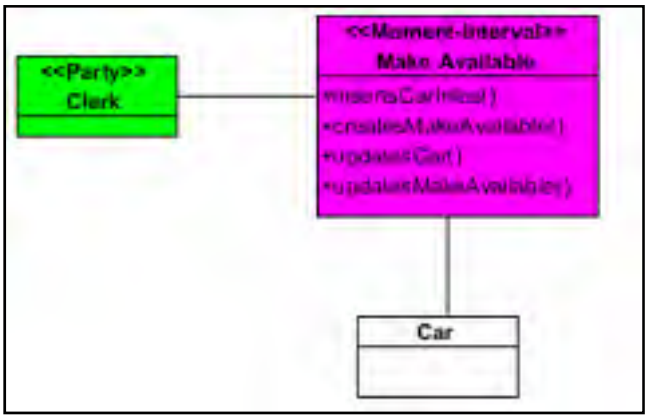


Figure 10.12 Diagramme de classe Make Available

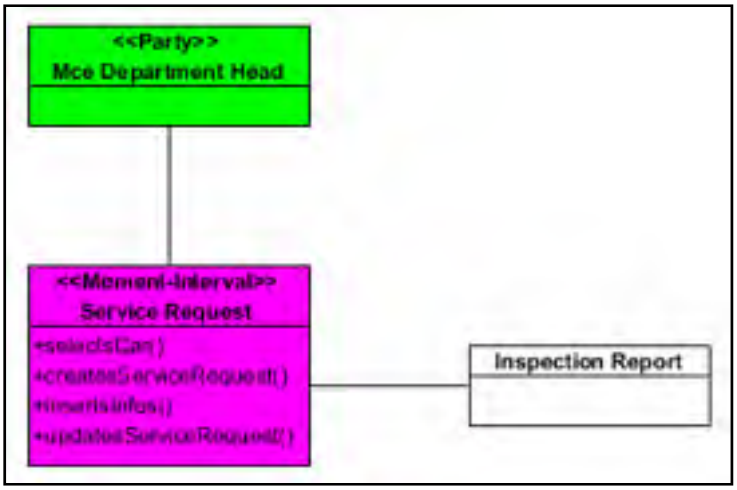


Figure 10.13 Diagramme de classe Service Request

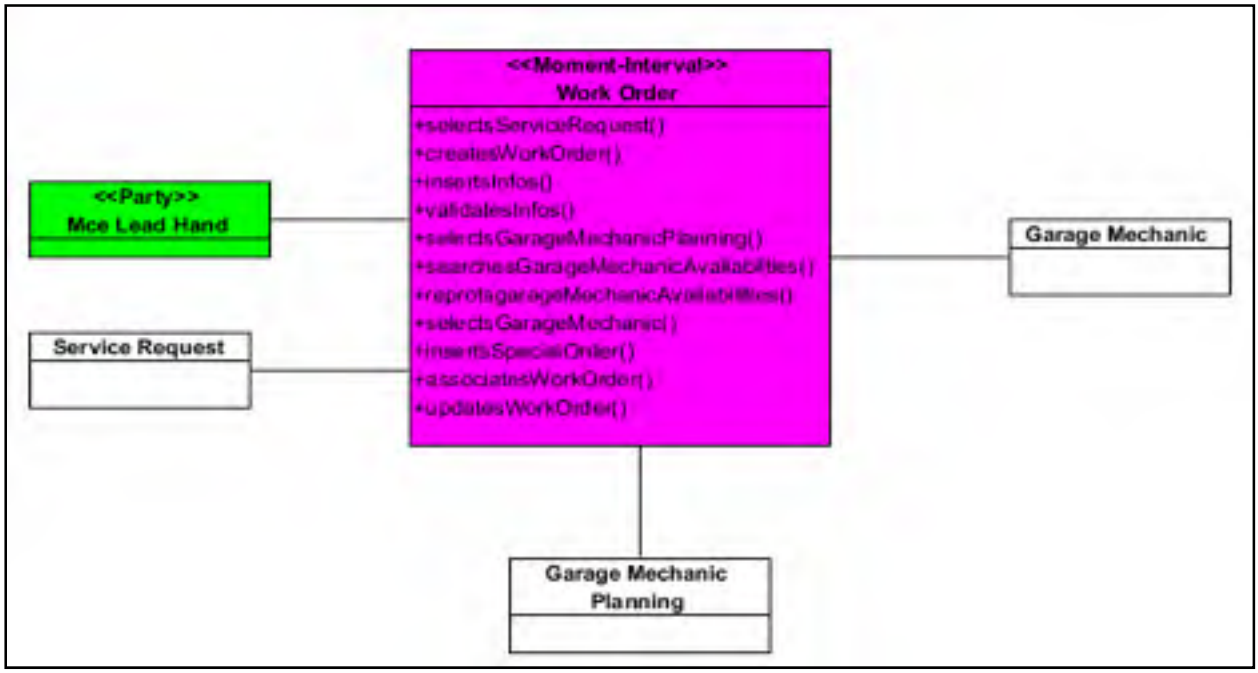


Figure 10.14 Diagramme de classe Work Order

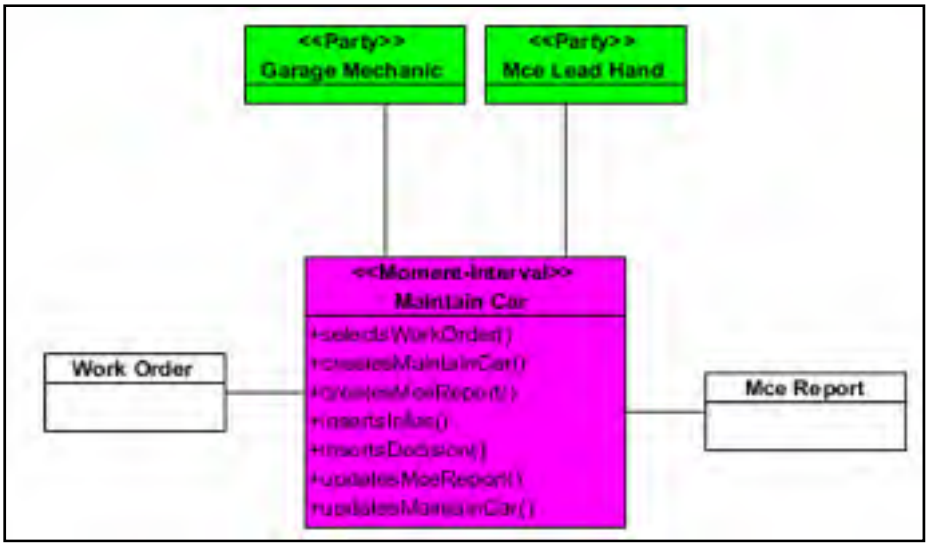


Figure 10.15 Diagramme de classe Maintain Car

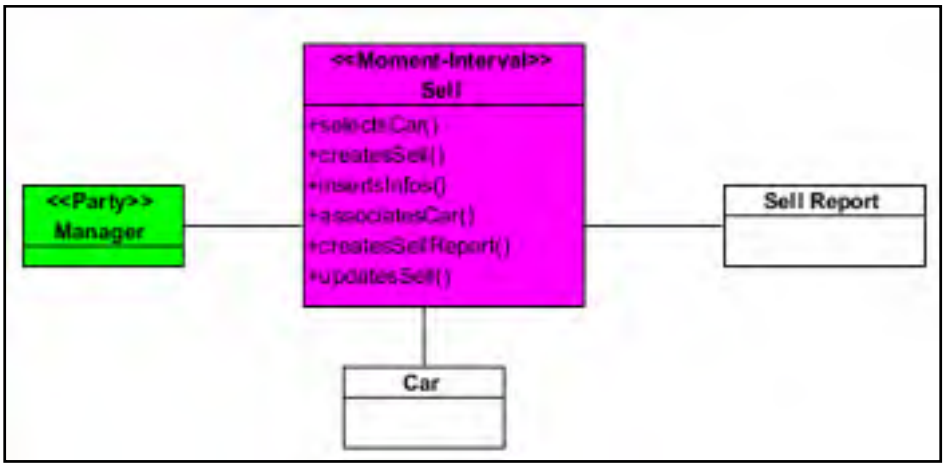


Figure 10.16 Diagramme de classe Sell

10.5.2 Expérimentation de l'étape procédurale

En appliquant l'étape procédurale (chapitre 9.1.2) sur l'ensemble des portions du diagramme de classe générées à partir de la première étape automatique, nous obtiendrons les diagrammes de classe suivants :

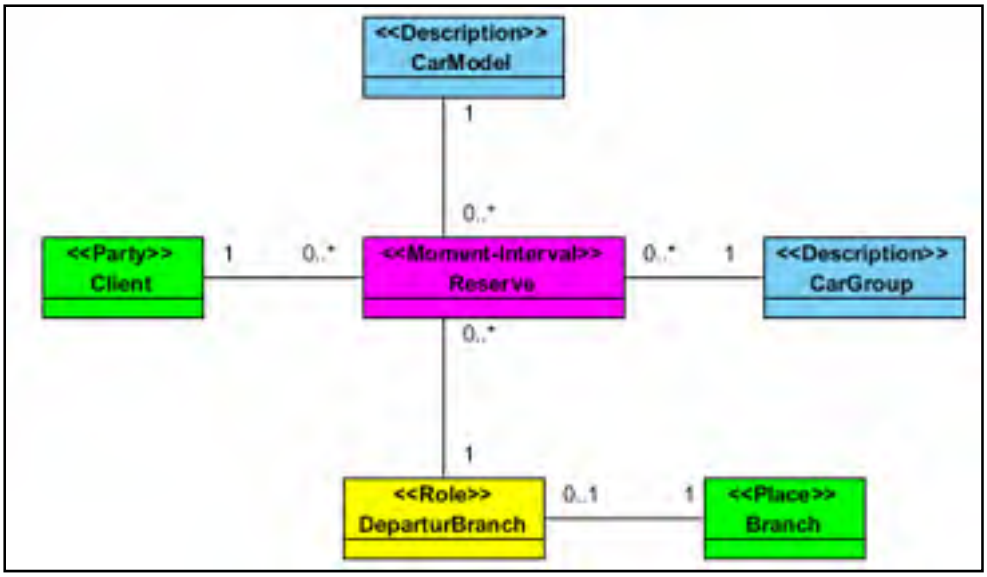


Figure 10.17 Diagramme de classe Reserve

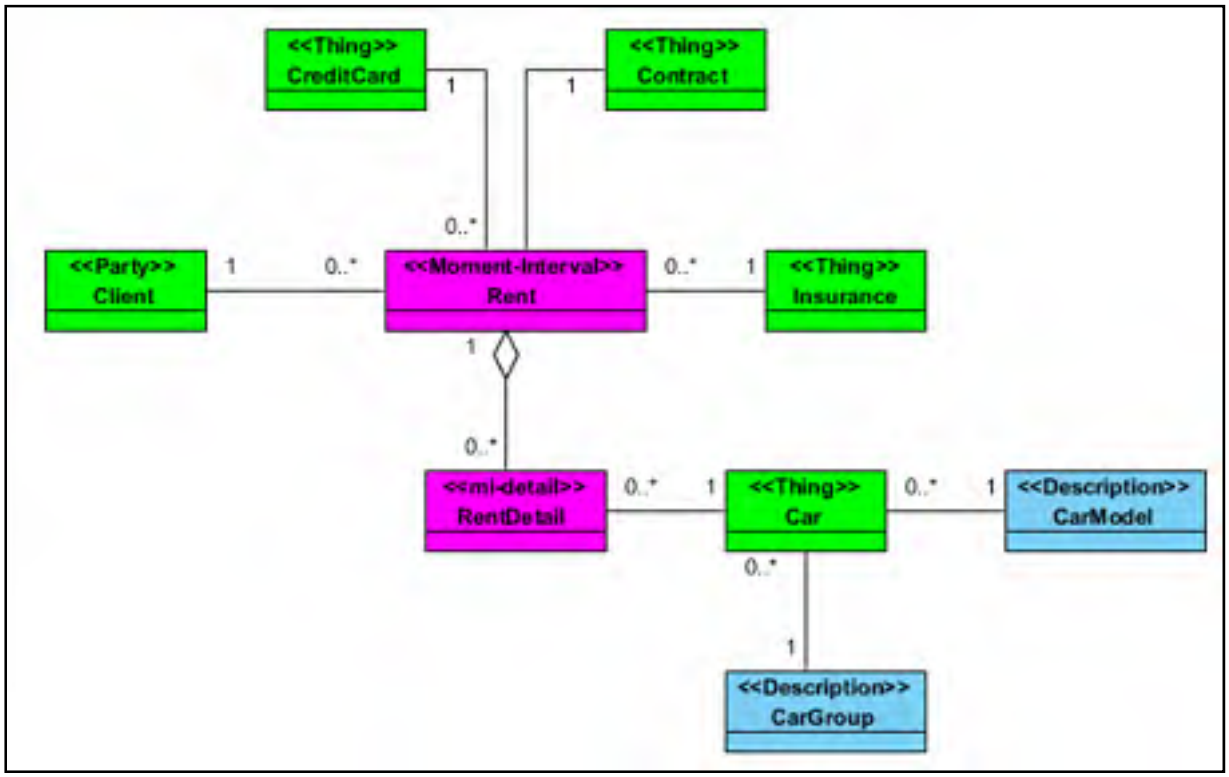


Figure 10.18 Diagramme de classe Rent

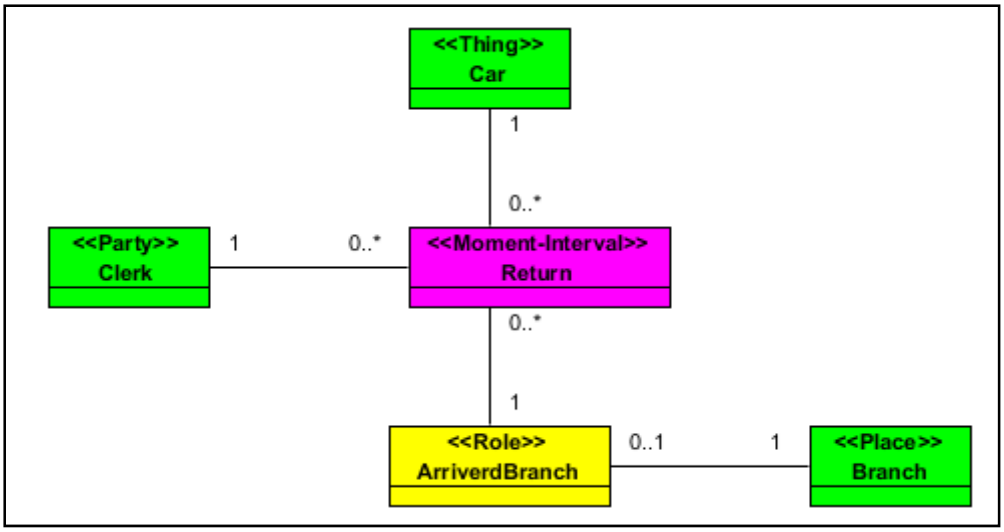


Figure 10.19 Diagramme de classe Return

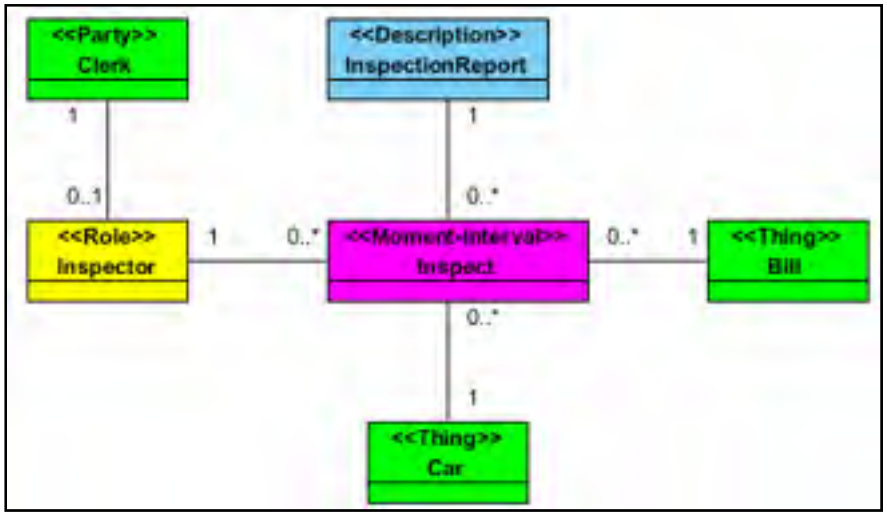


Figure 10.20 Diagramme de classe Inspect

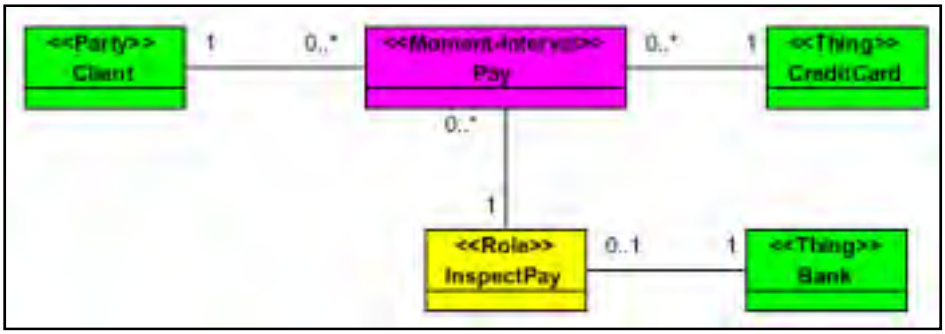


Figure 10.21 Diagramme de classe Pay

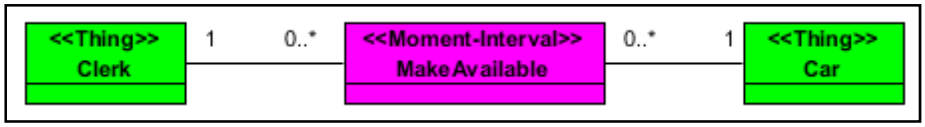


Figure 10.22 Diagramme de classe Make Available

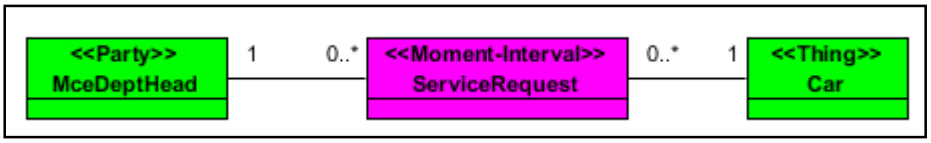


Figure 10.23 Diagramme de classe Service Request

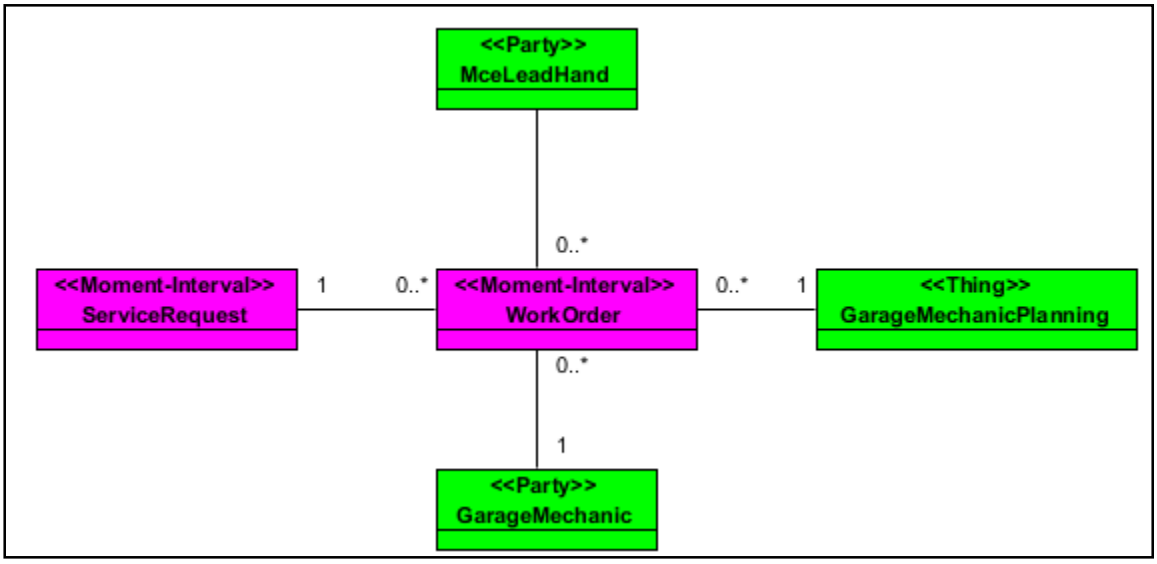


Figure 10.24 Diagramme de classe Work Order

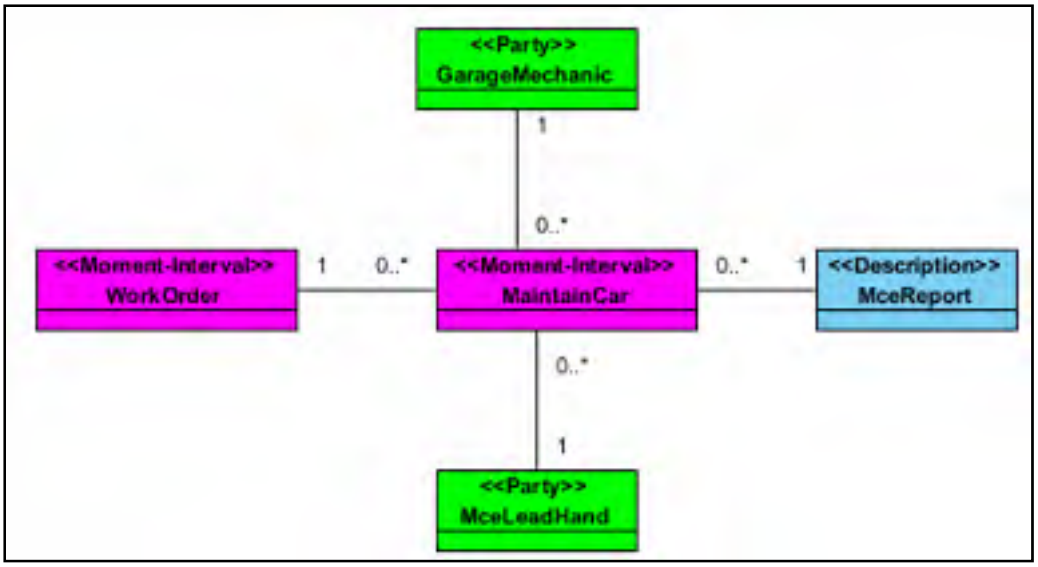


Figure 10.25 Diagramme de classe Maintain Car

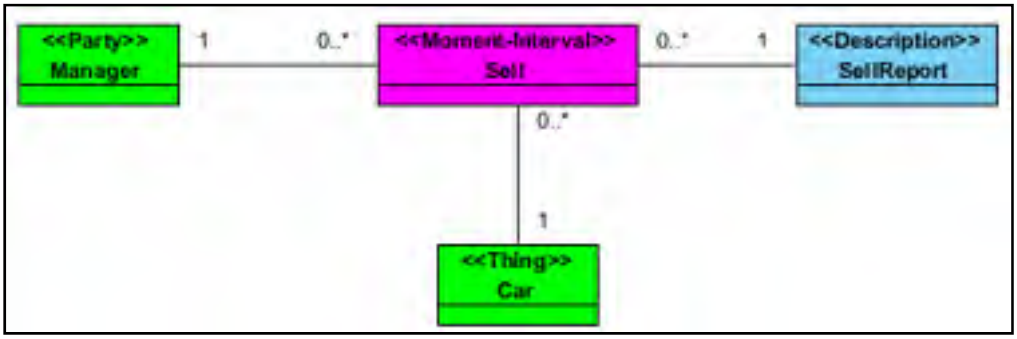


Figure 10.26 Diagramme de classe Sell

10.5.3 Expérimentation de l'étape 3

L'étape 3 consiste en l'application de deux règles R1 (LinkMI) et R2 (Merge class). Nous appliquons, respectivement, ces deux règles sur l'ensemble des portions de diagramme de classe. Nous avons décidé pour des raisons de simplicité de produire deux diagrammes de classe, un pour le service à la clientèle (Figure 10.27) et l'autre pour le service de maintenance d'une voiture (Figure 10.28).

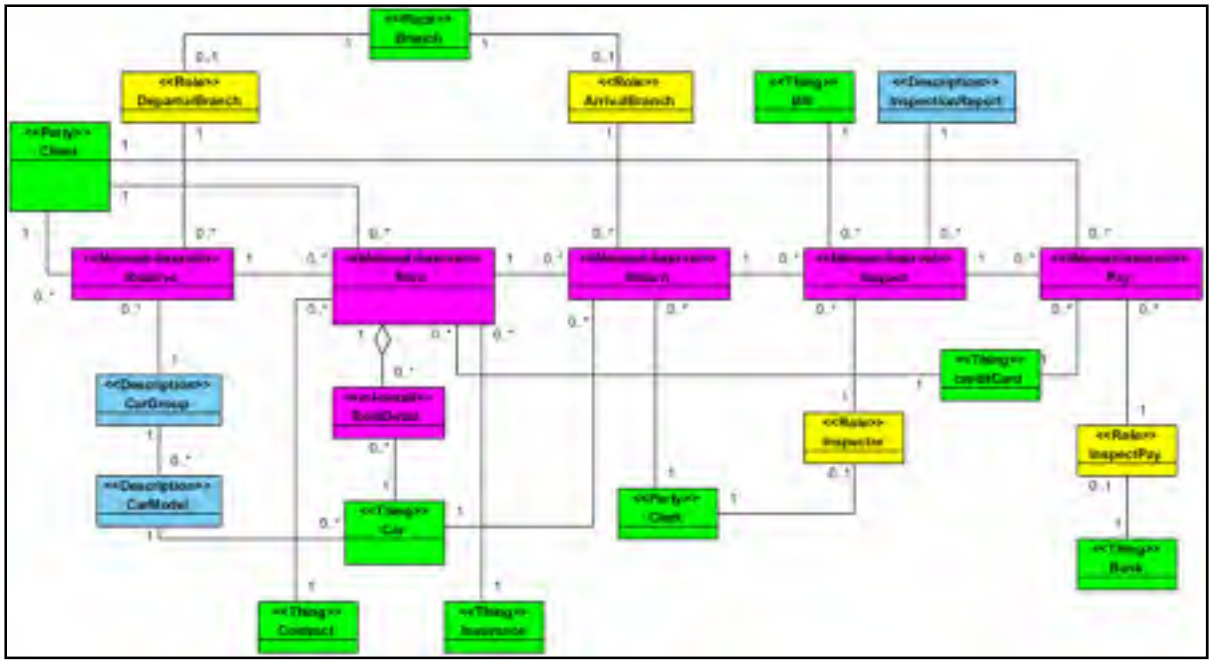


Figure 10.27 Diagramme de classe intégral du service à la clientèle

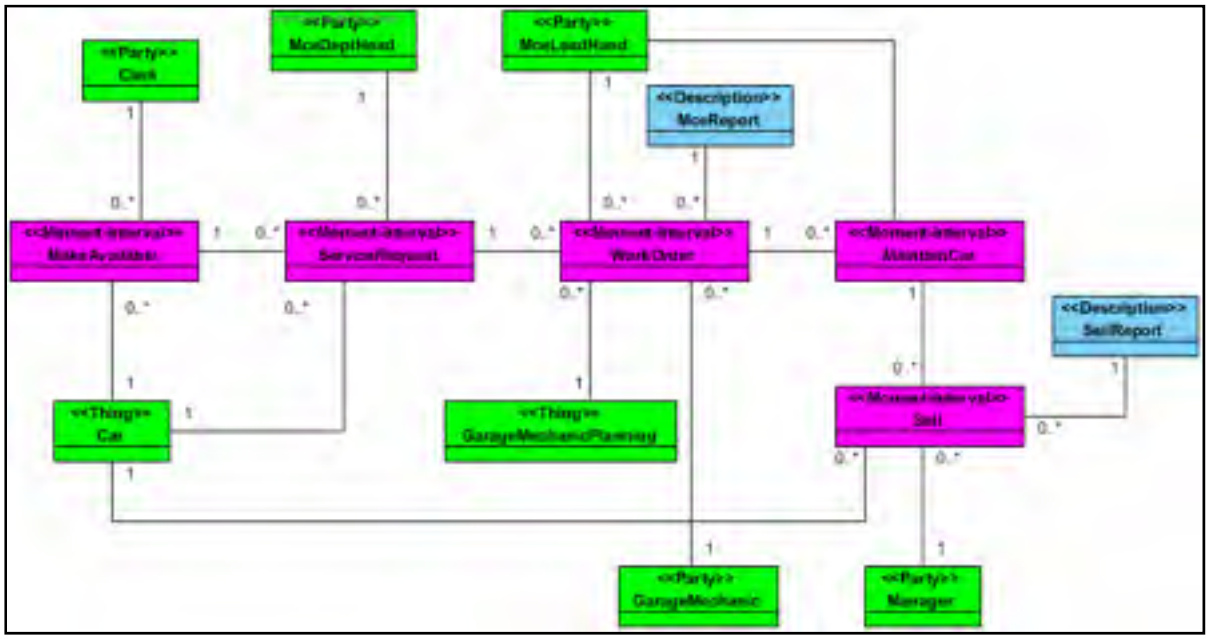


Figure 10.28 Diagramme de classe intégral de service maintenance

10.6 Discussion

Ce chapitre présente une expérimentation de la méthodologie de transformation du CIM en PIM en adoptant la méthode de l'étude de cas. Pour des raisons de simplification, nous nous sommes intéressés uniquement aux deux processus d'affaires Service à la clientèle et Maintenance d'une voiture. Cette décision n'affecte aucunement la validité de la méthodologie de transformation du CIM en PIM, plus particulièrement dans le cas où les processus d'affaires sont plus complexes. Dans notre recherche, la complexité d'un processus d'affaires se mesure uniquement par le nombre de noeuds de contrôle représentant des règles d'affaires. Les règles d'affaires du BPM n'ont pas d'influence directe sur la topologie des autres modèles (BCM, UCM, UC et le PIM), mais leur importance réside dans l'élaboration des liens entre les éléments de ces modèles, par exemple un lien entre deux cas d'utilisation ou deux classes.

La suite de l'expérimentation passe par l'application de l'ensemble des règles de transformation, de restriction et de contrôle présentées dans cette thèse.

CONCLUSION

L'objectif principal de cette thèse était de concevoir une méthodologie qui permette la transformation du CIM vers le PIM. L'approche MDA ne spécifie aucune méthodologie quant à l'élaboration de ces deux types de modèles ainsi que la manière de passer de l'un vers l'autre. Afin de réaliser cet objectif, la réalisation des sous-objectifs suivants a été nécessaire :

1) Définition de l'architecture du CIM.
2) Définition de l'architecture du PIM.
3) Définition de la technique de construction du modèle des processus d'affaires.
4) Définition de l'architecture du modèle de composants ainsi que les règles de transformation permettant de transformer le modèle de processus d'affaires en modèles de composants d'affaires.
5) Définition de la technique de transformation du modèle de processus d'affaires en modèles de cas d'utilisation.
6) Définition du gabarit permettant de rédiger les détails de chaque cas d'utilisation.
7) Définition du modèle permettant de modéliser un cas d'utilisation visuel ainsi que les règles de transformation permettant de transformer un cas d'utilisation textuel en un cas d'utilisation visuel.
8) Définition des règles de transformation permettant la transformation d'un cas d'utilisation en diagramme de classe d'analyse.
9) Définition d'une technique permettant de fusionner l'ensemble des portions de diagramme de classe, générée à partir de chaque cas d'utilisation, en un seul diagramme de classe d'analyse.
10) Expérimentation de la méthodologie de transformation du CIM en PIM par une étude de cas.

En résumé, dans cette thèse, une architecture du CIM, une architecture du PIM ainsi qu'une méthodologie de transformation du CIM en PIM sont proposées. Il s'agit en fait d'un cadre logiciel offrant la possibilité de constvruire le CIM à partir d'une composition de trois modèles (BMM, BPM et UCM) et d'une méthodologie de transformations successives de modèles. Celle-ci, fidèle aux principes de MDA, permet la transformation du CIM en PIM à l'aide de patrons d'analyse, de patrons de communication et des archétypes.

La motivation de ce travail de recherche était de réduire le fossé entre les activités d'analystes d'affaires et les architectes logiciels. Ce rapprochement se concrétise à l'aide de l'utilisation du même langage de modélisation UML2 et des règles de transformation permettant de passer d'un modèle à l'autre.

Contributions

Plusieurs contributions sont issues de cette thèse. Premièrement, le chapitre 3 a proposé une architecture du CIM composée de trois modèles interreliés soit le BMM, le BPM et l'UCM ainsi qu'une proposition de méthodologie permettant la création du CIM. La littérature fait état de l'absence de ce type d'architecture. L'originalité de cette architecture réside dans l'utilisation du BMM, adoptée comme standard par l'OMG en 2005, et à son intégration avec les modèles BPM et UCM. Cette architecture du CIM apporte une compréhension plus précise du modèle CIM en mettant en perspective l'ensemble des éléments de ce dernier ainsi que les liens entre eux. Cette contribution a permis de réaliser le sous-objectif 1.

Deuxièmement, le chapitre 4 a proposé une architecture du PIM basée sur les quatre archétypes de Coad et le DNC. L'utilisation de ces archétypes a été évaluée et validée par Coad ainsi que par d'autres chercheurs dans plusieurs projets de développement de logiciel. Cette architecture permet une compréhension plus précise du modèle PIM et contribue à définir les frontières entre le CIM et le PIM. De plus, une méthodologie permettant de construire le PIM est proposée et munie d'un cas d'étude. Cette contribution a permis de réaliser le sous-objectif 2.

Troisièmement, le chapitre 5 a proposé une technique de conception du BPM basée sur l'EBP. Un profil du BPM basé sur le métamodèle du diagramme d'activité UML2 a été aussi proposé. Celui-ci apporte plus de précision et de spécificité au diagramme d'activité en adaptant les éléments qui les constituent à la modélisation des processus d'affaires BPM. Cette contribution a permis de réaliser le sous-objectif 3.

Quatrièmement, le chapitre 6 a introduit le concept du diagramme de composants d'affaires (BCM) ainsi que le profil UML2 permettant de le modéliser. De plus, une technique permettant, à l'aide de règles de transformation, de transformer le BPM en BCM a été proposée. L'objectif du BCM réside dans la réutilisation des composants d'affaires dans d'autres applications logicielles. En d'autres mots, le BCM, une fois défini et validé, propose aux analystes d'affaires un catalogue de composants d'affaires prêt à utiliser. Cette contribution a permis de réaliser le sous-objectif 4.

Cinquièmement, le chapitre 7 a proposé une nouvelle technique de construction du modèle des cas d'utilisation (UCM) basée sur le diagramme d'activité UML2. Un profil UML2 de l'UCM ainsi que la technique permettant de transformer le BPM en UCM sont proposés. L'objectif de cette nouvelle technique consiste en l'utilisation d'un seul diagramme de modélisation capable, à la fois, de modéliser le BPM, le BCM et l'UCM. Ceci facilite les activités et réduit le temps d'apprentissage des analystes d'affaires. Cette contribution a permis de réaliser le sous-objectif 5.

Sixièmement, le chapitre 8 a proposé plusieurs autres contributions :

- Un gabarit de rédaction des cas d'utilisation sous format textuel renforcé par des patrons de communication structurant la syntaxe des interactions de l'acteur et du système, des verbes génériques limitant l'interprétation des verbes d'action de l'acteur et du système et des règles de restriction permettant d'informer et de guider l'analyste d'affaires sur les opérations non autorisées dans l'activité de rédaction du cas d'utilisation textuel.
- Une technique de rédaction de cas d'utilisation sous un format visuel à l'aide du diagramme d'activité UML2. Cette technique est accompagnée d'un profil UML2

permettant d'adapter le diagramme d'activité à la modélisation des cas d'utilisation ainsi que des règles de contrôle permettant le respect de sa syntaxe.

- De plus, des règles de transformation permettant de passer d'un cas d'utilisation textuel en un cas d'utilisation visuel sont proposées.

Cette contribution a permis de réaliser les deux sous-objectifs 6 et 7.

Septièmement, le chapitre 9 a proposé une technique de transformation du cas d'utilisation visuel en diagramme de classe d'analyse. Cette technique se compose de trois étapes :

1. La première consiste à appliquer des règles de transformation sur les cas d'utilisation visuels.
2. La deuxième à répondre à une série de questions afin de découvrir de nouveaux éléments du diagramme de classe.
3. La troisième consiste à fusionner l'ensemble des portions du diagramme de classe, générées à partir de chaque cas d'utilisation.

Cette contribution a permis de réaliser les deux sous-objectifs 8 et 9.

Huitièmement, dans le chapitre 10, une étude de cas a été réalisée dans le cadre de l'expérimentation de la méthodologie de transformation du CIM vers le PIM. Cette contribution a permis de réaliser le sous-objectif 10.

De plus, une preuve de concept permettant de valider partiellement l'automatisation des règles de transformation par des outils logiciels a été réalisée (Jean-François Charron et Benoit Lazzari, 2008). Cette preuve de concept concerne la transformation du cas d'utilisation visuel en diagramme de classe d'analyse en utilisant deux techniques distinctes :

- La première a été réalisée dans l'outil TogetherSoft de Borland en utilisant le langage de transformation de modèles QVT.
- La deuxième a été réalisée dans l'outil Rational Software Architect (RSA) d'IBM à l'aide de l'API java UML2.

En conclusion, nous considérons que l'objectif principal et les objectifs secondaires de cette recherche ont été accomplis.

Publications

Comme les contributions de cette thèse sont nombreuses, il n'a pas été possible de les soumettre en totalité à des conférences et des journaux avant la fin de la rédaction. En revanche, une partie des contributions de cette thèse mettant de l'avant des idées abordées dans le chapitre 3, le chapitre 4 et le chapitre 5 a été publiée dans les deux premiers articles. Le troisième article présente des résultats en relation avec le BMM. Celui-ci représente finalement une limite de recherche dans cette thèse et fera partie de travaux futurs.

1. Samir Kherraf, Alexandre Moïse, Éric Lefebvre, Witold Suryn, 'Towards a Structure for the Computation Independent Model', Proceedings of MDA and MTDD 2nd International Workshop on Model-Driven Architecture and Modeling Theory-Driven Development In conjunction with Evaluation of Novel Approaches to Software Engineering (ENASE), Athens, Greece, July 2010: p. 53-59.

2. Samir Kherraf, Eric Lefebvre, Witold Suryn, 'Transformation from CIM to PIM Using Patterns and Archetypes', 19th Australian Conference on Software Engineering. 2008, IEEE Computer Society, Perth, Australia. p. 338-346.

3. Samir Kherraf, Laila Cheikhi, Alain Abran, Witold Suryn, Eric Lefebvre, 'The Need to Evaluate Strategy and Tactics before the Software Development Process Begins', Journal of Software Engineering and Applications (JSEA), Scientific Research Publishing, July 2010. 3(7): p. 644-652. Disponible sur : http://www.scirp.org/journal/jsea/

Il est à noter qu'au même moment de la rédaction de cette thèse, cinq autres articles sont en cours de rédaction dont au moins deux seront soumis à des journaux.

Les limites de recherche

Bien que les contributions de cette thèse soient nombreuses, il n'a pas été possible d'apporter des solutions à toutes les problématiques de transformation du CIM en PIM. Les limites majeures de cette thèse se résument comme suit :

- L'utilisation du BMM dans le CIM n'a pas été suffisamment détaillée. Ainsi, les artefacts à produire durant la conception du BMM ne sont pas abordés. Ce travail dépasse le cadre de cette thèse.
- Bien que l'expérimentation de la méthodologie de transformation du CIM en PIM par le cas d'étude EU-RENT Car ait permis de montrer l'applicabilité de cette méthodologie, elle ne peut être suffisante pour démontrer complètement la validité et l'efficacité de cette méthodologie. D'autres expérimentations impliquant des praticiens et des chercheurs seront nécessaires afin de démontrer et de quantifier l'efficacité de la méthodologie proposée.
- En relation avec le point précèdent, il a été difficile de trouver des partenaires industriels familiers avec l'approche MDA étant disponible pour expérimenter cette méthodologie.

Avenues futures de recherche

Cette section présente quelques avenues de recherche pouvant être empruntées suite à cette thèse. Certaines de ces avenues sont associées aux limites identifiées à la section précédente.

La première avenue de recherche porte sur l'amélioration de l'architecture du CIM. Cette avenue se divise en trois voies :

1. La première consiste à améliorer le BMM de manière à ajouter des indicateurs de mesure des performances, des critères de qualité et des artefacts à produire.
2. La deuxième voie consiste à améliorer la méthodologie de création du CIM pour la rendre plus précise. En effet, la méthodologie de création du CIM proposée dans cette thèse a été créée sur la base d'une combinaison des étapes de création du BMM et des bonnes pratiques de création du BPM et du l'UCM sans l'avoir testée dans des cas pratiques. Elle pourrait donc faire l'objet d'expérimentations sur des projets de différentes tailles.
3. La troisième voie porte sur la création d'un seul profil UML du CIM capable de combiner les profils du BMM, du BPM et de l'UCM en un seul et unique profil. Celui-ci améliorerait potentiellement les relations et la traçabilité entre les différents éléments

constituant le CIM. Une piste de recherche serait d'étudier le standard *The Open Group Architecture Framework* (TOGAF) version 9 dont le BMM est supportée.

La deuxième avenue de recherche, qui se divise en deux voies, porte sur l'architecture du PIM :

1. La première voie consiste à créer des PIMs pour différents domaines d'affaires dont les règles et les pratiques d'affaires sont stables et maîtrisées. Pour ce faire, il serait intéressant de s'inspirer des 75 DNC (Domain Neutral Component) génériques proposés par Coad (Peter Coad, Eric Lefebvre et Jeff De Luca, 1999). Cette voie serait pertinente dans la composition et l'intégration des applications logicielles notamment dans la composition des modules d'un ERP.
2. Allant de pair avec la première voie, la deuxième voie porte sur la réingénierie de ces PIMs. Une fois ces PIMs définis et catégorisés, il serait intéressant d'en déduire les BPM correspondant.

La troisième avenue de recherche porte sur la validation de la méthodologie de transformation du CIM en PIM. Il serait intéressant d'évaluer cette méthodologie à l'aide d'une étude empirique, à large échelle, impliquant des industriels et des chercheurs universitaires spécialisés dans ce champ de recherche. Les résultats et les rétroactions de cette étude seraient utiles pour l'amélioration de cette méthodologie de transformation.

La quatrième avenue de recherche porte sur la création d'un outil logiciel capable de supporter, à la fois la création du CIM et sa transformation en PIM. Cette thèse présente les outils nécessaires à cet effet, à savoir l'architecture du CIM, l'architecture du PIM ainsi que les règles de transformation.

La cinquième avenue de recherche porte sur l'application des résultats de cette thèse sur les *Service Oriented Architecture* (SOA). Berkem (Birol Berkem, 2008a) introduit cette avenue en proposant d'aligner les éléments du BMM notamment la vision d'affaires, les buts, les stratégies, les tactiques et les règles d'affaires avec les composants du SOA.

ANNEXE I

CAS D'ÉTUDE : EU-RENT CAR RENTALS

Acknowledgement :

Model Systems, Ltd., developed this case study along with several other organizations participating in the Business Rules Group (www.businessrulesgroup.org). This text was taken as is from the "Appendix D" of the paper "Defining Business Rules ~ What Are They Really?" produced by the Business Rules Group. Other formats in which this paper is available are described in Defining Business Rules -- What Are They Really? (Abstract & Table of Contents)

EU-Rent Car Rentals

EU-Rent is a car rental company owned by EU-Corporation. It is one of three businesses - the other two being hotels and an airline - that each has its own business and IT systems, but with a shared customer base. Many of the car rental customers also fly with EU-Fly and stay at EU-Stay hotels.

EU-Rent business

EU-Rent has 1000 branches in towns in several countries. At each branch cars, classified by car group, are available for rental. Each branch has a manager and booking clerks who handle rentals.

Rentals

Most rentals are by advance reservation; the rental period and the car group are specified at the time of reservation. EU-Rent will also accept immediate ("walk-in") rentals, if cars are available.

At the end of each day cars are assigned to reservations for the following day. If more cars have been requested than are available in a group at a branch, the branch manager may ask other branches if they have cars they can transfer to him.

Returns

Cars rented from one branch of EU-Rent may be returned to a different branch. The renting branch must ensure that the car has been returned to some branch at the end of the rental period. If a car is returned to a branch other than the one that rented it, ownership of the car is assigned to the new branch.

Servicing

EU-Rent also has service depots, each serving several branches. Cars may be booked for maintenance at any time provided that the service depot has capacity on the day in question. For simplicity, only one booking per car per day is allowed. A rental or service may cover several days.

Customers

A customer can have several reservations but only one car rented at a time. EU-Rent keeps records of customers, their rentals and bad experiences such as late return, problems with payment and damage to cars. This information is used to decide whether to approve a rental.

EU-Rent Business Rules

External constraints

- Each driver authorized to drive the car during a rental must have a valid driver's licence.
- Each driver authorized to drive the car during a rental must be insured to the level required by the law of each country that may be visited during the rental.
- Rented cars must meet local legal requirements for mechanical condition and emissions for each country that may be visited during the rental.
- Local tax must be collected (at the drop-off location) on the rental charge.

Rental reservation acceptance

- If a rental request does not specify a particular car group or model, the default is group A (the lowest-cost group).
- Reservations may be accepted only up to the capacity of the pick-up branch on the pick-up day.
- If the customer requesting the rental has been blacklisted, the rental must be refused.
- A customer may have multiple future reservations, but may have only one car at any time.

Car allocation for advance reservations

At the end of each working day, cars are allocated to rental requests due for pick-up the following working day. The basic rules are applied within a branch:

- Only cars that are physically present in EU-Rent branches may be assigned.
- If a specific model has been requested, a car of that model should be assigned if one is available. Otherwise, a car in the same group as the requested model should be assigned
- If no specific model has been requested, any car in the requested group may be assigned
- The end date of the rental must be before any scheduled booking of the assigned car for maintenance or transfer
- After all assignments within a group have been made, 10% of the group quota for the branch (or all the remaining cars in the group, whichever number is lower) must be reserved for the next day's walk-in rentals. Surplus capacity may be used for upgrades.
- If there are not sufficient cars in a group to meet demand, a one-group free upgrade may be given (i.e. a car of the next higher group may be assigned at the same rental rate) if there is capacity
- Customers in the loyalty incentive scheme have priority for free upgrades.

If demand cannot be satisfied within a branch under the basic rules, one of the 'exception' options may be selected:

- A car may be allocated from the capacity reserved for the next day's walk-ins.

- A 'bumped upgrade' may be made. (For example, if a group A car is needed and there is no capacity in group A or B, then a car allocated to a group B reservation may be replaced by a group C car, and the freed-up group B car allocated to the group A reservation.)
- A downgrade may be made. A "downgrade" is a car of a lower group.
- A car from another branch may be allocated, if there is a suitable car available and there is time to transfer it to the pick-up branch.
- A car due for return the next day may be allocated, if there will be time to prepare it for rental before the scheduled pick-up time.
- A car scheduled for service may be used, provided that the rental would not take the mileage more than 10% over the normal mileage for service.

If demand cannot be satisfied within a branch under the 'exception' rules, one of the 'in extremis' options may be selected:

- Pick-up may have to be delayed until a car is returned and prepared.
- A car may have to be rented from a competitor.

Walk-in rentals

- The end date of the rental must be before any scheduled booking of the assigned car for maintenance or transfer.
- If there are several available cars of the model or group requested, the one with the lowest mileage should be allocated.

Handover

- Each driver authorized to drive the car during a rental must be over 25 and have held a driver's license for at least one year.
- The credit card used to guarantee a rental must belong to one of the authorized drivers; and this driver must sign the rental contract. Other drivers must sign an 'additional drivers authorization' form.

- The driver who signs the rental agreement must not currently have a EU-Rent car on rental.
- Before releasing the car, a credit reservation equivalent to the estimated rental cost must be made against the guaranteeing credit card.
- The car must not be handed over to a driver who appears to be under the influence of alcohol or drugs.
- The driver must be physically able to drive the car safely - must not be too tall, too short or too fat; if disabled, must be able to operate the controls.
- The car must have been prepared -- cleaned, full tank of fuel, oil and water topped up, tires properly inflated.
- The car must have been checked for roadworthiness -- tire tread depth, brake pedal and hand brake lever travel, lights, exhaust leaks, windscreen wipers.

No-shows

- If an assigned car has not been picked up 90 minutes after the scheduled pick-up time, it may be released for walk-in rental, unless the rental has been guaranteed by credit card.
- If a rental has been guaranteed by credit card and the car has not been picked up by the end of the scheduled pick-up day, one day's rental is charged to the credit card and the car is released for use the following day.

Return from rental

- At the end of a rental, the customer may pay by cash, or by a credit card other than the one used to guarantee the rental.
- If a car is returned to a location other than the agreed drop-off branch, a drop-off penalty is charged.
- The car must be checked for wear (brakes, lights, tires, exhaust, wipers etc.) and damage, and repairs scheduled if necessary.
- If the car has been damaged during the rental and the customer is liable, the customer's credit card company must be notified of a pending charge.

Early returns

- If a car is returned early, the rental charge is calculated at the rate appropriate to the actual period of rental (e.g. daily rate rather than weekly).

Late returns

- If the car is returned late, an hourly charge is made up to 6 hours' delay; after 6 hours a whole day is charged.
- A customer may request a rental extension by phone -- the extension should be granted unless the car is scheduled for maintenance.
- If a car is not returned from rental by the end of the scheduled drop-off day and the customer has not arranged an extension, the customer should be contacted.
- If a car is three days overdue and the customer has not arranged an extension, insurance cover lapses and the police must be informed.

Car maintenance & repairs

- Each car must be serviced every three months or 10,000 kilometers, whichever occurs first.
- If there is a shortage of cars for rental, routine maintenance may be delayed by up to 10% of the time or distance interval (whichever was the basis for scheduling maintenance) to meet rental demand.
- Cars needing repairs (other than minor body scratches and dents) must not bus used for rentals.

Car purchase and sale

- Only cars on the authorized list can be purchased.
- Cars are to be sold when they reach one year old or 40,000 kilometers, whichever occurs first.

Car ownership

- A branch cannot refuse to accept a drop-off of a EU-Rent car, even if a one-way rental has not been authorised.
- When a car is dropped off at a branch other than the pick-up branch, the car's ownership (and, hence, responsibility for it) switches to the drop-off branch when the car is dropped off.
- When a transfer of a car is arranged between branches, the car's ownership switches to the 'receiving' branch when the car is picked up.
- In each car group, if a branch accumulates cars to take it more than 10% over its quota, it must reduce the number back to within 10% of quota by transferring cars to other branches or selling some cars.
- In each car group, if a branch loses cars to take it more than 10% below its quota, it must increase the number back to within 10% of quota by transferring cars from other branches or buying some cars.

Loyalty Program

- To join the Loyalty Program, a customer must have made 4 rentals within a year.
- Each paid rental in the program (including the 4 qualifying rentals) earns points that may be used to buy 'free rentals.'
- Only the basic rental cost of a free rental can be bought with points. Extras, such as insurance, fuel and taxes must be paid by cash or credit card.
- A free rental must be booked at least fourteen days before the pick-up date.
- Free rentals do not earn points.
- Unused points expire three years after the end of the year in which they were earned.

Examples of "rules for running the business"

(not really the same kind of rules as those above)

- Each branch must be set targets for performance -- numbers of rentals, utilization of cars, turnover, profit, customer satisfaction, etc.

- Where performance requirements conflict (e.g. profit vs customer satisfaction when a customer requests a reduction in charges after an unsatisfactory rental) heuristics must be provided to guide branch staff.
- Performance data must be captured.

If performance targets are not met, control action must be taken. Control action may include:

- Changing the resources at branches (e.g. numbers of cars, quotas of cars within each group, number of staff),
- Changing responsibilities (e.g. having transfers of cars managed by groups of branches, rather than by negotiation between individual branch managers),
- Changing operational guidance (e.g. what proportion of cars should be kept for walk-in rentals), but not external constraints (e.g. legal requirements) or company policies (e.g. rentals must be guaranteed by a credit card, a customer may have only one car at a time.

BIBLIOGRAPHIE

Aditya Agrawal, Tihamer Levendovszky, Jon Sprinkle, Feng Shi et Gabor Karsai. 2002. « Generative Programming via Graph Transformations in the Model Driven Architecture ». In *Workshop on Generative Techniques in the Context of Model Driven Architecture In OOPSLA 2002*.

Anneke Kleppe, Jos Warmer et Wim Bast. 2001. *MDA Explained, The Model-Driven Architecture: Practice and Promise*. Addison Wesley,.

Bente Anda, et Dag I.K. Sjøberg. 2003. « Applying use cases to design versus validate class diagrams - a controlled experiment using a professional modeling tool ». In *International Symposium on Empirical Software Engineering*. p. 50-60. IEEE Computer Society.

Birol Berkem. 2008a. « From The Business Motivation Model (BMM) To Service Oriented Architecture (SOA), ». *Journal of Object Technology (JOT),* vol. 7, n^{o} 8, p. 57-70.

Birol Berkem. 2008b. « HOW TO ALIGN IT TO CHANGES on a Goal-Driven Service Oriented Architecture (GD-SOA) USING UML, MDA AND BMM (1)? ». *Goobiz*. < http://www.goobiz.com/How_to_align_IT_using_UML_and_according_to_BMM.htm >.

Boris Shishkov. 2002. « Business Engineering Building Blocks ». In *9th Doctoral Consortium on Advanced Inf. Systems Eng. (CAiSE)*.

Boris Shishkov, et Jan L. G. Dietz. 2004. « Design of Software Applications Using Generic Business Components ». In *Proceedings of the Proceedings of the 37th Annual Hawaii International Conference on System Sciences (HICSS'04) - Track 9 - Volume 9*. IEEE Computer Society.

Brian S. Mitchell. 2003. « A heuristic approach to solving the software clustering problem ». In *International Conference on Software Maintenance ICSM* p. 285-288.

Butler, G., P. Grogono et F. Khendek. 1997. « A Z specification of use cases: a preliminary report ». In *Software Engineering Conference, 1997. Asia Pacific ... and International Computer Science Conference 1997. APSEC '97 and ICSC '97. Proceedings*. (2-5 Dec 1997), p. 505-506.

Carl Gustav. 1981. *The Archetypes and The Collective Unconscious*. Princeton University Press.

Charles J. Puccia, et Richard Levins. 1985. *Qualitative Modeling of Complex Systems*. Harvard University Press, Cambridge MA.

Chris Raistrick, Paul Francis, John Wright, Colin Carter et Ian Wilkie. 2004. *Model Driven Architecture with Executable UML*. Cambridge.

Cockburn, Alistair. 2000. *Writing Effective Use Cases* Addison-Wesley Professional

Czarnecki, K., et S. Helsen. 2006. « Feature-based survey of model transformation approaches ». *IBM Syst. J.,* vol. 45, n° 3, p. 621-645.

David A. Taylor. 1995. *Business Engineering with Object Technology* John Wiley & Sons.

David S. Frankel. 2003. *Model Driven Architecture: Applying MDA to Enterprise Computing*. Wiley.

David V. Kerner. 1979. « Business information characterization study ». *SIGMIS Database,* vol. 10, n° 4, p. 10-17.

Dijkman, Remco M., et Stef M.M. Joosten. 2002. *Deriving Use Case Diagrams from Business Process Models*. University of Twente, Faculty of Computer Science.

E. Insfran, O. Pastor et R. Wieringa. 2002. « Requirements Engineering-Based Conceptual Modeling ». *Requirements Engineering Journal - Springer-Verlag,* vol. 7, p. 61-72.

Emmanuel Renaux. 2004. « Définition d'une démarche de conception de systèmes à base de composants ». Lille, Université des Sciences et Technologies de Lille.

Eric Lefebvre. 2005. « Buildong Platform-Independent Models with Business Archetypes and Patterns ». In *Montreal Conference on eTechnologies*

Erich Gamma, Richard Helm, Ralph Johnson et John M. Vlissides. 1994. *Design Patterns: Elements of Reusable Object-Oriented Software*. Addison-Wesley Professional.

Fowler Martin. 1996. *Analysis Patterns*. Addison-Wesley.

Freddy Allilaire, Jean Bézivin, Frédéric Jouault et Ivan Kurtev. 2006. « ATL: Eclipse Support for Model Transformation ». In *Proceeding of the Eclipse Technology eXchange Workshop (eTX) of the European Conference on Object-Oriented Programming (ECOOP)*.

Ganesan, R., et S. Sengupta. 2001. « O2BC: a technique for the design of component-based applications ». In *Technology of Object-Oriented Languages and Systems, 2001. TOOLS 39. 39th International Conference and Exhibition on*. p. 46-55.

Geert Jan Bex, Sebastian Maneth et Frank Neven. 2002. « A formal model for an expressive fragment of XSLT ». *Information System,* vol. 27, n° 1, p. 21-39.

George Pohle, Peter Korsten et Shanker Ramamurthy. 2005. *Component business models: Making specialization real*. Coll. « IBM Institute for Business Value study ».

Grady Booch, James Rumbaugh et Ivar Jacobson. 2005. *Unified Modeling Language User Guide, The (2nd Edition)*. Coll. « Object Technology Series ».

Grieskamp, W., et M. Lepper. 2000. « Using use cases in Executable Z ». In *Formal Engineering Methods, 2000. ICFEM 2000. Third IEEE International Conference on*. (2000), p. 111-119.

Hemant Jain, Naresh Chalimeda, Navin Ivaturi et Balarama Reddy. 2001. « Business Component Identification - A Formal Approach ». In *Proceedings of the 5th IEEE International Conference on Enterprise Distributed Object Computing*. IEEE Computer Society.

Hikita, Toshiya, et Masao J. Matsumoto. 2001. « Business Process Modeling Based on the Ontology and First-Order Logic ». In *ICEIS*.

Ibrahim, M., et R. Ahmad. « Class Diagram Extraction from Textual Requirements Using Natural Language Processing (NLP) Techniques ». In *Computer Research and Development, 2010 Second International Conference on*. (7-10 May 2010), p. 200-204.

Il-yeol Song and Kurt Yano, et Juan Trujillo. 2004. « A Taxonomic Class Modeling Methodology for Object-Oriented Analysis ». In. *Information Modeling Methods and Methodologies, Advanced Topics in Databases Series,*.

Interactive Objects Software GmbH. « ABB Uses ArcStyler to Web-Enable High-End Resources for Cost-Effective Access via the Intranet ». < http://www.omg.org/mda/mda_files/SuccessStory_ABB.pdf >.

Ivar Jacobson, Maria Ericsson et Agneta Jacobson. 1995. *The Object Advantage: Business Process Reengineering With Object Technology* Addison-Wesley.

Ivar Jacobson, Pan-Wei Ng. 2005. *Aspect-Oriented Software Development with Use Cases*, Addison-wesley. Coll. « Object Technology series ». Upper Saddle River: Pearson Education.

Ivar Jacobson, et Pan-Wei Ng. 2005. *Aspect-Oriented Software Development with Use Cases*. Coll. « Object Technology Series ». Addison-Wesley.

Ivar Jacobson et al. 1992. *Object Oriented Software Engineering: A Use Case Driven Approach*. Addison-Wesley.

Jack Greenfield, Keith Short. 2004. « Software factories, assembling applications with patterns,models, frameworks and tools ».

Jacobson, Ivar. 1992. *Object-Oriented Software Engineering: A Use Case Driven Approach*.

Jacobson, Ivar, Grady Booch et James Rumbaugh. 1999. *The Unified Software Development Process*. Coll. « Object Technology Series ». Addison-Wesley.

Janis Osis, Erika Asnina et Andrejs Grav. 2008. « Computation Independent Representation of the Problem Domain in MDA ». *e-Informatica Software Engineering Journal,* vol. 2, n^{o} 1.

Janis Osis, et Uldis Donins. 2010. « Platform Independent Model Development by Means of Topological Class Diagrams ». In *Workshop MDA & MTDD, International Conference on Evaluation of Novel Approaches to Software Engineering*. ENASE.

Jean-François Charron, et Benoit Lazzari. 2008. *Preuve de concept d'un programme de transformation du diagramme d'activité en diagramme de classe.* Coll. « Département de génie logiciel et TI ». Montreal: Université du Quebec - École de technologie supérieure.

Jean-Marie Van Der Maren. 1996. *Méthodes des recherches pour l'éducation*. De Boeck.

Jean Bézivin, Mireille Blay, Mokrane Bouzhegoub, Jacky Estublier, Jean-Marie Favre, Sébastien Gérard et Jean Marc Jézéquel. 2005. *Rapport de Synthèse de l'AS CNRS sur le MDA (Model Driven Architecture)*. Centre National de la Recherche Scientifique CNRS. < http://www-adele.imag.fr/mda/as/ >.

Jean Bézivin, et Olivier Gerbé. 2001. « Towards a Precise Definition of the OMG/MDA Framework. ». In *16th IEEE international Conference on Automated Software Engineering* (Washington, DC).

Jim Amsden. 2008. « Capturing requirements with Business Motivation Model, IBM Rational RequisitePro, and IBM Rational Software Modeler ». *DeveloperWorks, IBM*. < http://www.ibm.com/developerworks/rational/library/08/0401_amsden/ >.

Jim Arlow, et Ila Neustadt. 2004. *Enterprise Patterns and MDA: Building Better Software with Archetype Patterns and UML*. Addison Wesley.

Joaquin Miller, et Jishnu Mukerji. 2003. *MDA Guide Version 1.0.1*. OMG.

Johann Eder, et Walter Liebhart. 1996. « Workflow Recovery ». In *Proceedings of the First IFCIS International Conference on Cooperative Information Systems*. IEEE Computer Society.

John, Hunt. 2003. *Guide to the Unified Process Featuring UML, Java and Design Patterns*, 2. Springer.

Jorgensen, Paul, David Fernandez, Al Fischer, Michael Greco, Bradly Hussey, Steven Kuchta, Hong Li, Steven Overkamp, Douglas Rodenberger et Richard VanderWal. 2002. *Has the Object-Oriented Paradigm Kept Its Promise?* . Department of Computer Science and Information Systems, Grand Valley State University, USA

Julio Cesar Sampaio do Prado Leite, et Ann Paula M. Franco. 2003. « A Strategy for Conceptual Model Acquisition ». In *PROCEEDINGS OF THE FIRST INTERNATIONAL SYMPOSIUM ON REQUIREMENTS ENGINEERING*. IEEE Computer Society Press.

Karen M. Gardner, Alexander R. Rush, Michael Crist, Robert Konitzer et Bobbin Teegarden. 1998. *Cognitive Patterns : Problem-Solving Frameworks for Object Technology: Advances in Object Technology (SIGS: Managing Object Technology)*. Cambridge University Press.

Kent, Stuart. 2002. « Model Driven Engineering ». In *Proceedings of the Third International Conference on Integrated Formal Methods*. Springer-Verlag.

Kirikova, Marite, Anita Finke et Janis Grundspenkis. 2010. « What Is CIM: An Information System Perspective ». In *Advances in Databases and Information Systems*. p. 169-176. < http://dx.doi.org/10.1007/978-3-642-12082-4_22 >.

Kruchten Philippe. 2000. *The Rational Unified Process-An Introduction*, 2. Addison-Wesley.

Krzysztof Czarnecki, et Simon Helsen. 2003. « Classification of Model Transformation Approaches ». In *OOPSLA*

Krzysztof Czarnecki, et Simon Helsen. 2006. « Feature-based survey of model transformation approaches ». *IBM Syst. J.,* vol. 45, n° 3, p. 621-645.

Kurt Bittner, et Ian Spence. 2002. *Use Case Modeling*. Addison-Wesley Professional.

Larman, Craig. 2004. *Applying UML and Patterns*, 3. Prentice Hall.

Lee, Jong Kook, Seung Jae Seung, Soo Dong Kim, Woo Hyun et Dong Han Han. 2001. « Component Identification Method with Coupling and Cohesion ». In *Proceedings of the Eighth Asia-Pacific on Software Engineering Conference*. IEEE Computer Society.

Lefebvre Éric. 1996. « Améliorer les méthodes de planification informatique: une approche pluraliste ». Université Grenoble II.

Ludewig, Jochen. 2003. « Models in software engineering – an introduction ». *Software and Systems Modeling,* vol. 2, n° 1, p. 5-14.

Makrygiannis, Nickolas. 1998. « Toward mass-customized information systems ». In *Proceedings of the first component user's conference on Component-based software engineering*. (Munich, Germany). Cambridge University Press.

Mancoridis, S., B. S. Mitchell, Y. Chen et E. R. Gansner. 1999. « Bunch: a clustering tool for the recovery and maintenance of software system structures ». In *Software Maintenance, 1999. (ICSM '99) Proceedings. IEEE International Conference on*. p. 50-59.

Mark Lee. 2000. « Model-based reasoning: a principled approach for software engineering ». *Software - Concepts and Tools,* vol. 19, n° 4.

Markus Völter, Thomas Stahl, Jorn Bettin, Arno Haase et Simon Helsen. 2006. *Model-Driven Software Development: Technology, Engineering, Management*. Wiley.

Marlon Dumas, et Arthur H. M. Ter Hofstede. 2001. « UML activity diagrams as a workflow specification language ». In *UML conference*. sous la dir. de Verlag, Springer, p. 76-90.

Martin Fowler, et Kendall Scott. 1999. *UML Distilled: A Brief Guide to the Standard Object Modeling Language* 2. Addison-Wesley Professional.

Matthias Biehl, Chen DeJiu et Martin Törngren. 2010. « Integrating safety analysis into the model-based development toolchain of automotive embedded systems ». In *Conference on Languages, compilers, and tools for embedded systems* Vol. 45, p. 125-132.

Matthias Riebisch, et Michael Hübner. 2008. « Refinement and Formalization of SemiFormal Use Case Descriptions, Ilmenau ». < http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.83.8092 >.

Mellor Stephen J, Scott Kendall, Uhl Axel et Weise Dirk. 2004. *MDA distilled: principles of model-driven architecture*. Addison-Wesley Professional.

Mens Tom, et Van Gorp Pieter. 2006. « A Taxonomy of Model Transformation ». *Electronic Notes in Theoretical Computer Science,* vol. 152, p. 125-142.

Mohagheghi, Parastoo, Vegard Dehlen et Tor Neple. 2009. « Definitions and approaches to model quality in model-based software development - A review of literature ». *Information and Software Technology,* vol. 51, n° 12, p. 1646-1669.

Narayan Debnath, María Carmen Leonardi, María Virginia Mauco, Germán Montejano et Daniel Riesco. 2008. « Improving Model Driven Architecture with Requirements Models ». In *Information Technology: New Generations, 2008. ITNG 2008. Fifth International Conference on*. (7-9 April 2008), p. 21-26.

Niels Streekmann, Ulrike Steffens, Claus M¨obus et Hilke Garbe. 2006. « Model-driven integration of business information systems ». *Softwaretechnik-Trends,* vol. 26, p. 9-13.

Noam Chomsky. 1971. *Deep Structure, Surface Structure and Semantic Interpretation*. Steinberg and Jacobovits.

Object Management Group. 2008. *Query/View/Transformation Specification version 1.0*. OMG.

Object Management Group. 2010. « Success Stories ». < http://www.omg.org/mda/products_success.htm >.

Object Management Group. 2011. *Meta Object Facility (MOF) 2.0 Query/View/Transformation Specification. Version 1.1* OMG.

Oksana Nikiforova, et Natalya Pavlova. 2008. « Development of the Tool for Generation of UML Class Diagram from Two-Hemisphere Model ». In *The Third International Conference on Software Engineering Advances*.

Olfa Djebbi. 2004. *MDA : Vers l'industrialisation de construction d'applications réparties*. LIP6 (Laboratoire d'informatique Paris 6).

OMG. 2004. *Meta Object Facility (MOF) 2.0 Core Specification*. Object Management Group.

OMG. 2007. *Business Process Modeling Notation Specification version 1.1*. Object Management Group.

OMG. 2008. *Business Process Definition MetaModel (BPDM), Process Definitions,*. Object Management Group.

OMG. 2009-02-02. *UML 2.0 Superstructure*. OMG.

OMG. 2010a. *UML v2.3 - Infrastructure*. Object Management Group.

OMG. 2010b. *UML v2.3 - Superstructure*. Object Management Group.

OMG. 2011. « UML Resource Page ». < http://www.uml.org/ >.

OMG. September 2007 *Business Motivation Model (BMM) Specification* Object Management Group.

Oscar Pastor, Emilio Insfrán, Vicente Pelechano, José Romero et José Merseguer. 1997. « OO-METHOD: An OO Software Production Environment Combining Conventional and Formal Methods ». In *Proceedings of the 9th International Conference on Advanced Information Systems Engineering*. Springer-Verlag.

Oscar Pastor, Sergio España, Jose I. Panach et Natalie Aquino. 2008. « Model-Driven Development: piecing together the MDA jigsaw ». *Informatik-Spektrum. Special issue on Modelling,* vol. 31, n° 5, p. 394-407.

Osis Janis, Asnina Erika et Grave Andrejs. 2009. « Formal Problem Domain Modeling within MDA ». In *Software and Data Technologies*. p. 387-398. Springer Berlin Heidelberg. < http://dx.doi.org/10.1007/978-3-540-88655-6_29 >.

Parastoo Mohagheghi, et Vegard Dehlen. 2008. « Where is the Proof? - A Review of Experiences from Applying MDE in Industry ». In *4th European Conference on Model Driven Architecture Foundations and Applications (ECMDA'08),*. Vol. 5095, p. 432-443.

Patrick Coquillard, et David R. C. Hill. 1997. *Modélisation et Simulation d'Ecosystèmes : des modèles déterministes aux simulations à événements discrets*. Coll. « Recherche en Ecologie ». Masson, Paris.

Peter Coad, Eric Lefebvre et Jeff De Luca. 1999. *Java Modeling In Color With UML: Enterprise Components and Process*. Prentice Hall PTR.

Peter Coad, Eric Lefebvre, Jeff De Luca. 1999. *Java Modeling In Color With UML: Enterprise Components and Process*. Prentice Hall PTR.

Reynaldo Giganto, et Tony Smith. 2008. « Derivation of Classes from Use Cases Automatically Generated by a Three-Level Sentence Processing Algorithm ». In *Proceedings of the Third International Conference on Systems*. IEEE Computer Society.

Richard Hubert. 2001. *Convergent Architecture: Building Model Driven J2EE Systems with UML*. Wiley.

Rolland Colette, et Achour Camille Ben. 1998. « Guiding the construction of textual use case specifications ». *Data & Knowledge Engineering,* vol. 25, n° 1-2, p. 125-160.

Roussev, B. 2003. « Generating OCL specifications and class diagrams from use cases: a Newtonian approach ». In *Proceedings of the 36th Annual Hawaii International Conference on* sous la dir. de Sciences, System.

Samir Kherraf, Alexandre Moïse, Éric Lefebvre et Witold Suryn. 2010. « Towards a Structure for the Computation Independent Model ». In *2nd International Workshop on Model-Driven Architecture and Modeling Theory-Driven Development In conjunction with Evaluation of Novel Approaches to Software Engineering (ENASE)*. (Athens, Greece), sous la dir. de ENASE, p. 53 - 59. ENASE.

Samir Kherraf, Eric Lefebvre et Witold Suryn. 2008. « Transformation from CIM to PIM Using Patterns and Archetypes ». In *Proceedings of the 19th Australian Conference on Software Engineering*. sous la dir. de IEEE Computer Society.

Sang Duck, Lee, Yang Young Jong, Cho Fun Sook, Kim Soo Dong et Rhew Sung Yul. 1999. « COMO: a UML-based component development methodology ». In *Software Engineering Conference, 1999. (APSEC '99) Proceedings. Sixth Asia Pacific*. p. 54-61.

Scholtz, Jean, Shyam Chidamber, Robert Glass, Al Goerner, Mary Beth Rosson, Mike Stark et Iris Vessey. 1993. « Object-oriented programming: The promise and the reality ». *Journal of Systems and Software,* vol. 23, n° 2, p. 199-204.

Seidewitz, E. 2003. « What models mean ». *Software, IEEE,* vol. 20, n° 5, p. 26-32.

Selic, Bran. 2003. « The Pragmatics of Model-Driven Development ». *IEEE Softw.,* vol. 20, n° 5, p. 19-25.

Sengupta, S., et S. Bhattacharya. 2006. « Formalization of UML use case diagram-a Z notation based approach ». In *Computing & Informatics, 2006. ICOCI '06. International Conference on*. (6-8 June 2006), p. 1-6.

Sharma, V. S., S. Sarkar, K. Verma, A. Panayappan et A. Kass. 2009. « Extracting High-Level Functional Design from Software Requirements ». In *Software Engineering Conference, 2009. APSEC '09. Asia-Pacific*. (1-3 Dec. 2009), p. 35-42.

Stachowiak, Herbert. 1973. *Allgemeine Modelltheorie*. Coll. « Wien and New York ». Springer-Verlag.

Taylor David. 1995. *Business Engineering with Object Technology*. Wiley.

THE MIDDLEWARE COMPANY. 2003. *Model Driven Development for J2EE Utilizing a Model Driven Architecture (MDA) Approach : Productivity Analysis*. Object Management Group.

Thomas Stahl, Markus Völter, Jorn Bettin, Arno Haase et Simon Helsen. 2006. *Model-Driven Software Development: Technology, Engineering, Management* Wiley.

Tim Weilkiens, et Bernd Oestereich. 2008. *UML 2 Certification Guide: Fundamental & Intermediate Exams*. Morgan Kaufmann.

Toacy Cavalcante de Oliveira, Ivan Mathias Filho, Carlos Jos et Pereira de Lucena. Using XML and Frameworks to Develop Information Systems. « 2001 ». In *ICEIS*. p. 571-577.

Tratt, Laurence. 2005. « Model transformations and tool integration ». *Journal of Software and Systems Modelling,* vol. 4, n° 2.

Uday Apte, Chetan S. Sankar, Meru Thakur et Joel E. Turner. 1990. « Reusability-based strategy for development of information systems: implementation experience of a bank ». *MIS Q.,* vol. 14, n° 4, p. 421-433.

Walter M. Carlson. 1979. « Business information analysis and integration technique (BIAIT): the new horizon ». *SIGMIS Database,* vol. 10, n° 4, p. 3-9.

WFMC. 1999. *Workflow Management Coalition, Interface 1: Process Definition Interchange Process Model* <http://www.wfmc.org/ >.

Wil van der Aalst, et Arthur ter Hofstede. 1999. « Workflow Patterns ». < http://www.workflowpatterns.com/ >.

Withall, Stephen. 2007. *Software Requirement Patterns (Best Practices)*, 1. Microsoft Press, 384 p.

Xiaohua Zhou, et Nan Zhou. 2008. « Auto-generation of Class Diagram from Free-text Functional Specifications and Domain Ontology Abstract ». < http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.99.2062 >.

Yves Chiricota, Fabien Jourdan et Guy Melançon. 2003. « Software components capture using graph clustering ». In *11th IEEE International Workshop on Program Comprehension,*. p. 217-226. EEE Computer Society

Zoé Drey, Cyril Faucher, Franck Fleurey, Vincent Mahé et Didier Vojtisek. 2010. *Kermeta language, Reference manual*.