Langage contrôlé pour la spécification des règles métier dans le contexte de la modélisation des systèmes d'information

Paul Brillant Feuto Njonko

UNIVERSITÉ DE FRANCHE-COMTÉ

ÉCOLE DOCTORALE «LANGAGES, ESPACES, TEMPS, SOCIETÉS»

Thèse en vue de l'obtention du titre de docteur en

SCIENCES DU LANGAGE
SPÉCIALITÉ : TRAITEMENT AUTOMATIQUE DES LANGUES

LANGAGE CONTRÔLÉ POUR LA SPÉCIFICATION DES RÈGLES MÉTIER DANS LE CONTEXTE DE LA MODÉLISATION DES SYSTÈMES D'INFORMATION

Vol.1

Présentée et soutenue publiquement par

Paul Brillant FEUTO NJONKO

Le 25 novembre 2014

Sous la direction de Madame le Professeur Sylviane CARDEY

Membres de jury :

Sylviane CARDEY, Professeur à l'université de Franche-Comté

Walid EL ABED, Docteur à Global Data Excellence, Genève

Peter GREENFIELD, Professeur à l'université de Franche-Comté

Ruslan MITKOV, Professeur à l'université de Wolverhampton, Angleterre, Rapporteur

Christophe ROCHE, Professeur à l'université de Savoie, Chambéry

Patrick SAINT-DIZIER, Directeur de recherche à l'IRIT, Toulouse, Rapporteur

Dédicace

Je dédie cette thèse

À ma très chère Grand-Mère NGOUNOUO Jeanne

Remerciements

J'adresse mes sincères remerciements à Madame le Professeur Sylviane CARDEY sans qui ce travail n'aurait point vu le jour. Je lui exprime ma profonde gratitude pour tout le soutien et les encouragements qu'elle m'a apportés pendant la réalisation de ce travail. Ses précieux conseils m'ont permis de transcender tous les problèmes que j'ai dû affronter.

J'adresse également mes sincères remerciements au Dr Peter GREENFIELD pour son soutien inconditionnel pendant la réalisation de ce travail et son entière disponibilité pour la relecture et les corrections de ce mémoire.

Je remercie les fondateurs de *Global Data Excellence* (GDE) le Dr Walid EL ABED et Monsieur Marc Vengadabady ainsi que toute la dynamique équipe de GDE pour leur franche collaboration et leur soutien pendant la réalisation de ce travail.

Je remercie tous les chercheurs et membres du Centre de recherche Lucien Tesnière pour le climat de convivialité qui a toujours régné entre nous depuis mon arrivée au Centre.

Je remercie toute ma famille, mes ami(e)s pour leurs encouragements pendant les moments difficiles.

À tous ceux et celles qui de près ou de loin ont rendu ce travail possible, je leur adresse mes sincères remerciements.

Résumé

Notre thèse s'inscrit dans le cadre des langages contrôlés pour le génie logiciel. Elle a pour but de faciliter l'adoption de l'approche par règles métier (ARM) par les entreprises en créant un langage contrôlé en vue d'aider à la spécification des règles métier par les experts métier. Notre solution va permettre de réduire la distance sémantique entre les experts métier et les experts système afin de répondre non seulement au besoin d'intercompréhension entre ces derniers mais aussi pour réaliser un transfert automatique de la description des règles métier vers les systèmes d'information (SI). Ce langage contrôlé que nous avons créé permettra d'assurer en plus la consistance et la traçabilité de ces règles avec leur implantation.

Durant ces dernières décennies, les règles métier ont été d'un intérêt considérable tant dans le monde industriel que celui de la recherche. En effet, les chercheurs et les professionnels se sont convaincus qu'elles constituent la composante la plus sensible aux changements liés à l'évolution du métier. Les règles métier sont considérées comme le véhicule idéal pour capturer la logique métier qui est à la base des opérations et des décisions dans une entreprise. Elles sont également au cœur des exigences fonctionnelles des SI et ont la capacité de les rendre flexibles, maintenables, évolutifs et entièrement alignés avec les besoins et l'évolution du métier.

Toutefois, les approches traditionnelles de modélisation des SI ne permettent pas une représentation propre et centralisée des règles métier. Elles sont implicitement implémentées en dur à différents endroits dans les applications métier (interfaces utilisateurs, codes sources, bases de données, etc.) et ceci souvent de manière incohérente. Ainsi, elles sont inaccessibles aux décideurs que sont les experts métier (experts de santé, de finance, etc.) qui doivent de plus en plus être dépendants de l'informatique. Changer ou modifier une règle métier nécessite inévitablement l'intervention des informaticiens qui la plupart de temps ne sont pas conscients de la pertinence de la décision à portée de main. Ceci peut non seulement conduire à des erreurs, mais nécessiter un coût et un délai importants dû aux contraintes des plateformes technologiques. En conséquence, les entreprises ne sont pas flexibles et ne peuvent s'adapter rapidement et à moindre coût à l'évolution du métier. La solution consiste donc à séparer la logique métier (règles métier) de la logique système (applications métier). Dans la littérature, l'approche par règles métier et l'ingénierie dirigée par les modèles (IDM)

autour du *Model Driven Architecture* (MDA) de l'*Object Management Group* (OMG) sont deux approches éprouvées et complémentaires qui ont contribué à cette solution. Elles sont toutes deux à la quête de l'agilité, avec des leviers permettant une nette séparation entre la logique métier et la logique système ce qui permettrait aux entreprises de s'adapter très rapidement et à moindre coût aux changements tant au niveau du métier qu'au niveau des plateformes technologiques. L'ARM préconise l'externalisation des règles métier des SI et leur gestion de façon centralisée par les experts métier. Elles pourront ainsi être implantées dans les SI par l'intermédiaire des moteurs de règles. Le MDA à travers son architecture en couche permet aux experts système de modéliser ces règles métier à différents niveaux d'abstractions pour différentes parties prenantes (experts métier, experts système, etc.) et machines. Cependant, la gestion de l'ambigüité du langage métier (naturel) pour permettre l'expression formelle des règles métier par les experts métier et leurs transformations vers des modèles appropriés aux moteurs de règles reste un enjeu majeur et constitue une problématique pour ces deux approches. Une solution prometteuse est la définition d'un langage contrôlé qui est un bon médiateur entre le langage naturel des experts métier et le langage formel des moteurs de règles.

Ainsi, cette thèse s'inscrit dans cette problématique et vise à proposer un langage contrôlé appelé RuleCNL pour la spécification formelle des règles métier par les experts métier et un moteur de transformations automatiques vers des modèles formels à destination de moteurs de règles. Notre idée est motivée par la publication par l'OMG dans le contexte de l'ARM du standard SBVR (*Semantics of Business Vocabulary and Business Rules*) qui définit un méta-modèle sémantique basée sur la logique formelle pour spécifier la structure du sens du vocabulaire et des règles métier exprimés en langage naturel. Ainsi, les modèles formels générés par notre moteur de transformations sont des modèles sémantiques SBVR appelés formulations sémantiques. Ces modèles sémantiques pourront non seulement être échangés entre différents outils et entreprises, mais pourront aussi être mappés vers différents autres modèles exécutables ou des artefacts logiciels grâce aux mécanismes de transformation de modèles du MDA.

Mots-clés : règles métier, systèmes d'information, ARM, IDM, MDA, langage contrôlé, langage naturel, SBVR, formulations sémantiques, transformations de modèles.

Abstract

Our thesis focuses on controlled natural languages (CNL) for software engineering. It aims at facilitating the adoption of the business rule approach (BRA) by companies by creating a CNL in order to help business experts in the specification of their business rules. Our solution will allow reducing the semantic gap between business experts and system experts to meet not only the need for mutual understanding between them but also to achieve an automatic transfer of the description of business rules to information systems (IS). The CNL that we have created will also ensure the consistency and the traceability of these rules together with their implementation.

In recent decades, business rules have received much attention within both the business community and also the academic community. Indeed, professionals and researchers are convinced that they are the most sensitive component in respect of business changes. Business rules are considered as the ideal vehicle for capturing the business logic which is the foundation for business operations and decisions. They are also at the heart of the functional requirements of IS because of their ability to make business applications flexible as well as being amenable to change, scalable and fully aligned with the business needs and the latters' evolution. However, traditional IS modeling approaches do not allow for the proper and centralized representation of business rules. These latter have been hard-coded and scattered over different places in business applications (user interfaces, source code, databases, etc.) and often inconsistently so. Thus, they are inaccessible to decision makers such as domain experts (healthcare experts, finance experts, etc.) who must be increasingly dependent on system experts. Changing or modifying a business rule inevitably requires the intervention of system experts who most of the time are not aware of the relevance of the decision at hand. This can be not only prone to errors, but requires a significant cost and delay due to the constraints of technology platforms. As a result, companies are not flexible and cannot keep pace with the changing business environment. The solution is to separate the business logic (business rules) from that of the system logic (business applications). In the literature, the BRA and the model-driven engineering (MDE) around the Model Driven Architecture (MDA) of the Object Management Group (OMG) are two proven and complementary approaches which have contributed to this solution. They both seek for agility together with

leverage allowing a clear separation between the business logic and the system logic which would thus allow companies to adapt rapidly and inexpensively to changes at the business level as well as at the level of technology platforms. The BRA claims that all business rules should be collected, removed from IS and centrally managed by business experts. The business rules can thus be implemented in IS through rule engines. The MDA through its layered architecture allows system experts to model business rules at different abstraction levels for different stakeholders (business experts, system experts, etc.) and machines. However, managing the ambiguity of business language (natural) to allow a formal expression of business rules by business experts and their mappings to appropriate models for rule engines remains a real challenge and is problematic for both approaches (BRA and MDA). A promising solution is the definition of a CNL that is a good mediator between the natural language of business experts and the formal language of rule engines.

Thus, this thesis, which is concerned with this problem area, aims at providing a CNL, which we call RuleCNL, for the formal specification of business rules by business experts and an engine for the automatic transformation of these to formal rule models for rule engines. Our idea is motivated by the publication by the OMG in the context of the BRA of the SBVR (Semantics of Business Vocabulary and Business Rules) standard which defines a semantic metamodel based on formal logic for specifying the structure of the meaning of business vocabulary and rules expressed in natural language. Thus, formal models generated by our transformation engine are semantic models called SBVR semantic formulations. These semantic models can not only be exchanged between different companies and tools, but can also be mapped to different other executable models or other software artifacts by means of MDA model transformation mechanisms.

Keywords: business rules**,** information systems, BRA, MDE, MDA, CNL, natural language, SBVR, semantic formulations, model transformations.

Table des matières

Table des figures

Page

Liste des tableaux

Page

Liste des abréviations, sigles et acronymes

ACE : Attempto Controlled English

AGL : Atelier de génie logiciel

ANTLR : ANother Tool for Language Recognition

ARM : Approche par règles métier

ATL : Atlas Transformation Language

BMM : Business Motivation Model

BRG : Business Rule Group

BRMS : Business Rule Management System

B2B : Business to Business

B2C : Business to Customer

CANLP : Computer Aided Natural Language Processing

CFG : Context-free grammar

CIM : Computation Independent Model

CNL : Controlled Natural Language

CWM : Common Waherouse Metamodel

DSL : Domain Specific Language

DTD : Document Type Definition

EMF : Eclipse Modeling Framework

ETL : Extract Transform Load

GAO : Government Accountability Office

IA : Intelligence artificielle

IDM : Ingénierie dirigée par les modèles

MDA : Model Driven Architecture

MDE : Model Driven Engineering

MOF : Meta Object Facility

MSLA : Micro-Systemic Linguistic Analysis

LC : Langage contrôlé

LiSe : Linguistique et Sécurité

OCL : Object Constraint Language

OMG : Object Management Group

ORM : Object Relational Model

OWL : Ontology Web Langage

PIM : Platform Independent Model

PRR : Production Rule Representation

PSM : Platform Specific Model

R2ML : REWERSE Rule Markup Language

RDF : Resource Description Framework

REWERSE : Reasoning on the Web with Rules and Semantics

RIF : Rule Interchange Format

RuleCNL : Rule Controlled Natural Language

RUP : Rational Unified Process

SBVR : Semantics of Business Vocabulary and Business Rules

SI : Systèmes d'information

SQL : Structured Query Language

SWRL : Semantic Web Rule Language

TAL : Traitement automatique des langues

UML : Unified Modeling Language

UP : Unified Process

W3C : World Wide Web Consortium

XML : eXtensible Markup Language

XMI : XML Metadata Interchange

XP : eXtreme Programming

YACC : Yet Another Compiler Compiler

Chapitre 1

Introduction générale

Dans ce chapitre introductif, dans un premier temps, nous exposons le contexte et les motivations de notre recherche puis sont présentées, la problématique, les approches existantes ainsi que leurs limites, et enfin les contributions de la thèse et l'organisation du document.

1.1 Contexte et motivations

Dans le contexte économique actuel, les entreprises font face à de nombreuses pressions internes et externes (compétitivité, croissance, concurrence sur le marché, etc.) qui nécessitent des adaptations fréquentes de leur politique (logique) métier. Ceci nécessite bien souvent des changements ou des modifications à la fois dans leur processus métier et dans leurs systèmes d'information (SI). Toutefois, il est clair que les entreprises modernes qui résistent le plus à ces pressions sont celles qui s'adaptent le plus rapidement et à faible coût à ces nouveaux changements. Durant ces dernières décennies, les règles métier ont été d'un intérêt considérable tant dans le monde industriel que celui de la recherche. En effet, les chercheurs et les professionnels se sont convaincus qu'elles constituent la composante la plus sensible aux changements liés au métier (Ross, 1997 ; Gottesdiener, 1997 ; Date, 2000). Les règles métier jouent un rôle important dans la définition de la sémantique d'une entreprise et sont considérées comme le véhicule idéal pour capturer la logique métier. Elles influencent ou guident le comportement du métier et soutiennent la politique métier afin de répondre aux événements environnementaux. En somme, elles représentent le principal moyen par lequel une entreprise définit sa politique, cordonne ses opérations, oriente ses prises de décisions afin d'atteindre ses objectifs comme le dit un expert des règles métier dans la citation suivante :

"*Business rules are the ultimate levers with which business management is able to guide and control the business. In fact, the business's rules are the means by which an organization implements competitive strategy, promotes policy, and complies with legal obligations*"
(Von Halle et Goldberg, 2006, p.4).

Les règles métier sont également au cœur des exigences fonctionnelles (Olivé, 2007) des SI et ont la capacité de les rendre flexibles, maintenables, évolutifs et entièrement alignés avec les besoins et l'évolution du métier. Elles peuvent être considérées comme le plus important lien entre le métier et les SI (Bajec et Krisper, 2005). Pour ces raisons, les règles métier nécessitent d'être traitées soigneusement et explicitement dans les entreprises pendant la modélisation des SI afin de garantir l'évolution du métier et l'agilité des SI (Ram et Khatri, 2005 ; Bajec et Krisper, 2005).

Cependant, les approches traditionnelles de modélisation des SI (*Unified Process (UP)*, *Rational Unified Process (RUP)*, *eXtreme Programming (XP*), etc.) ne permettent pas une représentation propre et centralisée des règles métier. La plupart des règles métier ne sont pas explicitement modélisées pendant les phases d'analyse et de conception, mais sont plutôt implicitement implémentées en dur à différents endroits dans les applications métier (interfaces utilisateurs, code source, bases de données, processus métier, etc.) et ceci souvent de manière incohérente (Bultleris et Kapocius, 2002). Cette tendance a conduit à une migration des règles métier (la logique métier) dans les SI sous forme de ligne de code. Ceci rend ainsi les décideurs que sont les experts métier de plus en plus dépendants de l'informatique. Par contre, la logique métier est à la base des opérations et des décisions dans une entreprise. Cette tendance présente plusieurs inconvénients à savoir :

1) Les SI développés ne répondent pas toujours aux exigences et aux besoins du métier.

Des recherches indiquent que seulement 35% des SI développés s'achèvent à temps et respectent le budget initial (Johnson, 1995 ; 2007). En réalité, les approches traditionnelles de modélisation des SI commencent généralement à partir de la formulation et la description du problème à résoudre (exigences des besoins) dans un formalisme de modélisation comme UML (*Unified Modeling Language*) (Rumbaugh et al., 2004) et de balisage comme XML (*eXtensible Markup Language*) Schéma (Harold, 2001). À partir de là, les experts système (ingénieurs logiciels) transforment ces exigences en code en suivant un processus bien défini. Toutefois, ces formalismes ont été conçus à destination des ingénieurs logiciels, dont le but ultime est de concevoir des artefacts logiciels. Par conséquent, ils utilisent des concepts et des notations qui ne sont pas facilement compréhensibles par les experts métier (experts de santé, de finance, etc.) qui comprennent le domaine du problème réel et sont responsables de trouver les solutions. Visiblement, la principale difficulté se trouve en amont, c'est à dire pendant la

description du problème et les fonctionnalités attendues. Alors pendant cette phase, les experts métier impliqués dans le processus de développement initient des discussions avec des ingénieurs logiciels pour leur exprimer leurs règles métier en utilisant leur langage métier proche du langage naturel. Ce langage métier n'est pas assez formel pour être interprété de façon claire et sans ambigüités aucunes. Le langage naturel, bien qu'offrant un pouvoir expressif et communicationnel très riche, est aussi riche en constructions complexes et en ambigüités sémantiques. Ainsi, l'expression de ces règles métier dans n'importe quel langage naturel est inévitablement ambiguë et la plupart de temps incomplète, ce qui peut provoque des incohérences et ambigüités provenant d'erreurs d'interprétations qui conduisent à la mise en œuvre des SI qui ne répondent pas non seulement aux exigences du métier, mais qui sont aussi difficile à maintenir et à faire évoluer. À ce sujet, le *Government Accountability Office* (GAO) américain déclarait :

"*Ill-defined or incomplete requirements have been identified by many experts as a root cause of system failure. As a case in point, we recently reported that the initial deployment of a new Army system intended to improve depot operations was still not meeting user needs, and the Army expected to invest $1 billion to fully deploy the system. One reason that users had not been provided with the intended systems capability was a breakdown in requirements management*." (GAO, 2006).

Il existe une distance sémantique et linguistique entre les experts métier et les experts système. Ceci implique un énorme effort de la part des experts système pour comprendre le sens et les concepts qui enveloppent le discours des experts métier. Ainsi, réduire cette distance sémantique entre le langage métier et les langages formels utilisés dans le processus de développement des SI constitue un enjeu fondamental pour une méthodologie complète et efficace.

2) Les approches traditionnelles inhibent l'évolution de la politique métier et par conséquent des règles métier qui sont exposées à des changements rapides.

Les règles métier sont les actifs (composants) de l'entreprise les plus prompts aux changements qui peuvent être internes ou externes à l'entreprise. Ceci peut provenir de la politique de vente d'un produit qui change, l'arrivée d'un nouveau produit, d'une opération marketing, de la fidélisation des clients à travers des réductions, de la promulgation d'une nouvelle loi ou d'un nouveau standard (norme), d'un nouveau partenariat créé, de nouvelles

avancées technologiques à prendre en compte, etc. Ainsi, séparer les règles métier du code source des SI pour les rendre flexibles aux changements est essentiel, surtout lorsque l'environnement est multidevise, multinational et multiculturel. Toutefois, dans les entreprises réglementées (pharmaceutiques, finances, etc.) et même non réglementées, si les règles métier sont enfouies et dispersées à différents endroits dans les applications informatiques (Herbst et al., 1994), il sera toujours difficile pour un expert métier de connaître les règles qui sont implémentées, comment et où elles le sont, ce qui peut conduire à des implémentations redondantes et incohérentes de certaines règles. À cet égard, la logique métier peut contenir des contraintes et des règles qui ne sont pas explicitement connues dans l'entreprise ou encore, l'entreprise peut supposer des contraintes et des règles, qui ne sont pas implémentées. Ainsi, ceci rend la logique métier difficile à maintenir et à faire évoluer car les règles métier ne sont pas regroupées, documentées et traitées de façon centralisée (Bajec et Krisper, 2005). Changer ou modifier une règle métier nécessite inévitablement l'intervention des ingénieurs logiciels car elle est incompréhensible et donc inaccessible aux experts métiers qui comprennent le changement réel du domaine et sont responsables de trouver des solutions et de prendre des décisions. (Fu et al., 2004). Les experts système pourront ne pas être conscients de la pertinence de la décision à portée de main, ce qui peut non seulement conduire à des erreurs, mais nécessiter un coût et un délai importants dû aux contraintes des plateformes technologiques ou d‟exécutions. Par conséquent, les entreprises ne sont pas flexibles, elles ne peuvent pas suivre le rythme de l‟évolution du contexte commercial et ne peuvent pas réagir rapidement au moment opportun à de nouvelles opportunités du métier.

1.2 Problématique de la thèse

Au vu de ce qui précède, on peut se poser la question suivante :

Dans les méthodologies ou approches de modélisation des SI, comment séparer la logique métier (règles métier) et la logique système (applications métier) ?

Ainsi, on pourra restreindre le langage naturel des experts métier afin de leur permettre de pouvoir eux mêmes modéliser leur logique /modèle métier (vocabulaire et règles métier) de façon formelle pour ensuite les transformer vers des modèles formels. À partir de là, les ingénieurs logiciels ne s‟occuperont que de la logique applicative et des aspects liés aux plateformes d‟exécutions, ce qui permettra non seulement de réduire la distance sémantique

entre les experts métier et les experts système afin d'éviter des défaillances et des incohérences dans les SI liées aux ambigüités de communication en langage naturel, mais ceci permettra également aux règles métier d'être gérer par les experts métier comme des citoyens de première classe, implémentées dans les SI de manière à évoluer à la vitesse des contraintes de l'environnement métier, et non à la vitesse des contraintes technologiques.

Pour essayer de répondre à cette question, nous nous sommes intéressés à l'étude et à l'analyse des approches existantes dans la littérature. Cette problématique a suscité de nombreuses recherches dans le milieu universitaire et dans l'industrie. Des paradigmes et approches de modélisation des SI se sont succédés afin d'apporter des solutions. Dans la littérature, l'approche par règles métier (ARM) (BRG, 2003) et l'ingénierie dirigée par les modèles (IDM) ou le *Model Driven Architecture* (MDA) de l'*Object Management Group* (OMG)[1] (OMG, 2003a) sont deux approches éprouvées et complémentaires pour la modélisation des SI qui ont contribué dans cette direction.

1.2.1 Approche par règles métier

Un expert en SI expose très clairement dans la citation suivante sa vision de l'ARM comme suit :

"Years of experience with information system development have taught us two important lessons:

It takes far too long to turn a relatively simple set of requirements into a system that meets user needs, and the cost of converting existing applications to new technologies is prohibitive.

The factor underlying both of these problems is the amount of code it takes to build a system. If code is the problem, the only possible solution is to eliminate the coding by building systems directly from their specifications. That is what the [Business Rule Approach does]." (Huber, 1997).

L'approche par règles métier ou *Business Rule Approach* (BRA) proposée par le *Business Rule Group* (BRG)[2] est une méthodologie visant à apporter une solution pour la modélisation

[1] http://www.omg.org

[2] http://www.businessrulesgroup.org/bra.shtml

des SI orientés métier centrée sur les règles métier. Son principe de base est la séparation nette entre la logique métier (règles métier) de la logique système permettant aux experts métier de spécifier leurs règles métier dans un environnement zéro développement se basant sur le langage naturel (Diouf, 2007). Les règles métier sont au cœur de l'approche et sont considérées comme les concepts fondamentaux. Le plus grand avantage de l'approche est que les règles métier sont compréhensibles par les experts métier et peuvent également être utilisées directement dans les SI (Gawel et Skalna, 2012 ; Gottesdiener, 1997). Elles doivent être collectées et explicitement gérées dans une application centralisée appelée moteur de règles ou système de gestion de règles métier ou - *Business Rule Management System* (BRMS) (Von Halle, 2001 ; Butleris et Kapocius, 2002). Ainsi, elles pourront être gérées et maintenues par des experts métier et ensuite être implémentées de façon cohérente dans tous les processus métier et les systèmes d'information de l'entreprise par l'intermédiaire des moteurs de règles ce qui contribuera à les externaliser des processus métier et des systèmes d'information, permettant ainsi d'avoir un gain significatif en terme d'agilité face aux changements. Elles pourront constituer une entité de première classe de l'entreprise qui sera à la base des opérations et des décisions (Morgan, 2001) (cf. figure 1.1). Leur impact dans les processus métier et systèmes d'information peuvent être évalué et tracé. Quand une décision métier doit être prise (par exemple, savoir si on doit accepter ou rejeter une réclamation d'assurance), les règles métier sont des déclarations atomiques (indivisible sans perte d'information) de la logique métier qui sont évaluées par le moteur de règles pour déterminer le résultat de la décision.

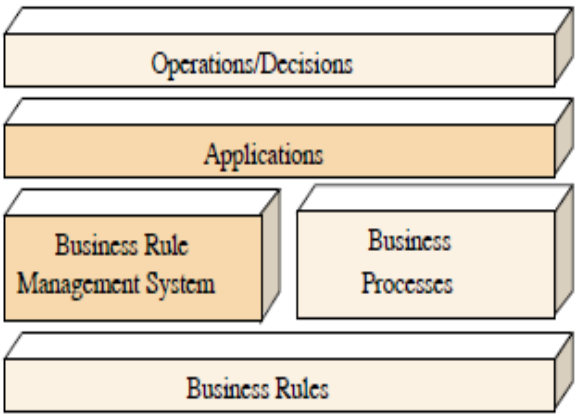


Figure 1.1 : Les règles métier comme la base des opérations/décisions

L'ARM est de plus en plus adoptée et utilisée dans les entreprises particulièrement dans le secteur des finances, des banques et des assurances. Sa mise en œuvre repose principalement sur l'utilisation d'un BRMS ou d'un moteur de règles qui doit permettre la spécification des règles métier, leur stockage dans un référentiel de règles, et leur exécution à la demande des applications métier. De plus en plus de BRMS existent aussi bien commerciaux que gratuits et/ou libres (*Drools, JRules, Jess*, etc.), avec plus ou moins de fonctionnalités facilitant ainsi la mise en œuvre de SI orientés règles métier[3]. La plupart de BRMS existants de nos jours proposent plutôt des langages de règles formels et techniques (Braye et al. 2006 ; Linehan, 2008) pour éviter toute ambigüité sémantique dans leur exécution.

Cependant, l'une des motivations clés de l'ARM est l'utilisation en amont du langage naturel pour l'expression, la création ou la vérification des règles par les experts métier, qui pour la plupart ne connaissent pas les formalismes de modélisation. Ainsi, dans un processus de mise en œuvre de l'ARM, les règles métier doivent être représentées dans différents langages pour différentes parties prenantes (experts métier, experts système et machines) allant du langage naturel des experts métier aux langages formels des BRMS pour exécution (Von Halle, 2001 ; Ram et Khatri, 2005 ; Gawel et Skalna, 2012). Pour cela, l'approche MDA (décrite dans la section suivante) a été proposée par l'OMG dans le contexte de l'ingénierie dirigée par les modèles (IDM). C'est une approche basée sur les modèles qui préconise la conception des modèles/langages à différents niveaux d'abstraction et le passage d'un niveau à un autre par des transformations automatiques ou semi-automatiques.

1.2.2 Ingénierie dirigée par les modèles autour du MDA

L'ingénierie dirigée par les modèles (IDM) ou (*Model Driven Engineering - MDE*) est apparue suite à une longue évolution (voir Section 3.2) dans le génie logiciel. Elle a considérablement contribuée à la maitrise de la complexité des SI distribués et la montée en abstraction grâce aux modèles. Dans l'ère IDM, le modèle est devenu le concept unificateur et est placé au centre du processus de modélisation des SI. Le principe de l'IDM est la montée en abstraction grâce aux modèles et permet aux ingénieurs logiciels de se soustraire aux détails techniques de l'implémentation pour se focaliser dans un premier temps sur des modèles métier indépendants de toutes plateformes d'exécutions. Ces modèles pourront être

[3] Les SI orientés règles métier sont implémentés suivant l'approche par règles métier.

ensuite transformés automatiquement ou semi automatiquement vers des modèles enrichis avec des détails d'implémentation. L'avantage est que les changements au niveau métier seront indépendants des plateformes d'exécutions et vice versa permettant ainsi à chacun d'évoluer à son propre rythme. Ainsi, les changements au niveau métier permettront de répondre aux besoins métier facilement adaptables par les experts métier sans intrusions technologiques et les changements technologiques permettront de tirer profit de nouvelles plateformes technologiques. L'adoption de l'IDM s'est fortement amplifiée avec le standard MDA en 2000 de l'OMG qui définit les modèles à trois niveaux d'abstractions comme le montre la figure 1.2.

1.2.2.1 Les modèles CIM (*Computational Independant Model*)

On a au premier niveau les modèles métier (*CIM*) (OMG, 2003a) où sont définis le vocabulaire, les règles métier et les exigences du système et ceci sans aucune considération informatique et détails d'implémentation. À ce niveau, les règles métier sont exprimées dans un langage métier qui peut être naturel ou visuel (présenté sous forme de diagrammes). Il s'agit des règles métier de haut niveau qui sont identifiées typiquement en fonction des objectifs qui définissent la vision de l'entreprise. Ces règles sont considérées comme des déclarations qui décrivent comment le métier a choisi de réaliser des objectifs et comment les politiques métier les plus importantes seront mises en œuvre.

1.2.2.2 Les modèles PIM (*Platform Independant Model*)

Au second niveau, nous avons les modèles d'analyses et de conceptions *(PIM*) (OMG, 2003a) qui sont des modèles informatiques indépendants de toutes plateformes d'exécutions. À ce niveau, le langage naturel n'est pas approprié pour exprimer les règles métier. Elles doivent être exprimées de façon structurée, formelle ou semi-formelle. La plupart sont exprimées dans un langage basé sur XML ou sur UML. Puisque les règles métier constituent la vraie essence des exigences des SI, c'est à ce niveau que se trouvent les activités les plus importantes qui sont effectuées au cours de la modélisation des SI.

1.2.2.3 Les modèles PSM (*Platform Specific Model*)

Enfin au niveau 3, nous avons des modèles d'implémentations et de code *(PSM)* (OMG, 2003a) qui sont des modèles spécifiques aux plateformes d'exécutions. À ce niveau, les règles métier sont exprimées dans un langage spécifique à un BRMS donné. À partir d'un modèle PSM, on peut générer du code dans une plateforme d'exécution donnée (ex : J2EE, .Net, etc.).

Ce découpage permet ainsi de séparer les spécifications fonctionnelles d'un SI des spécifications de son implémentation sur une plateforme d'exécution donnée et de déployer le même modèle sur plusieurs plateformes grâce à des projections standardisées (transformations de modèles). Ceci apporte un gain significatif en productivité et en portabilité des SI.

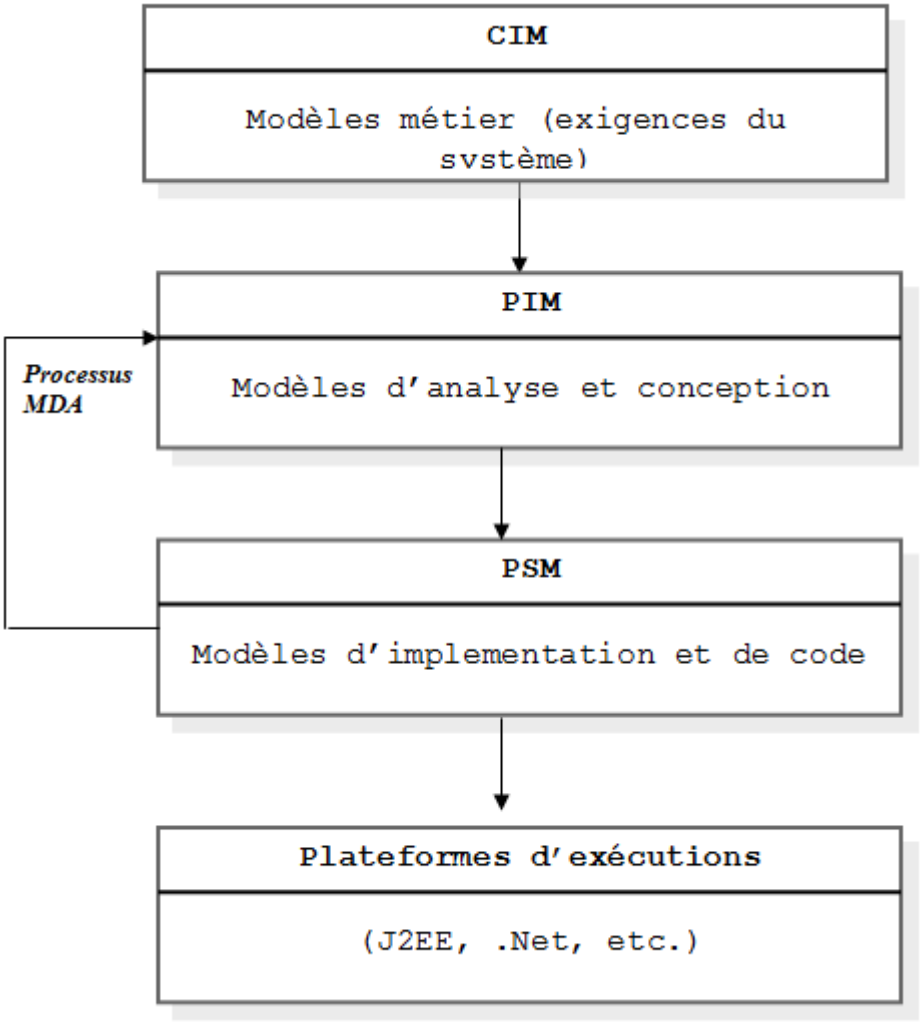


Figure 1.2 : Différents niveaux d'abstraction des modèles MDA

Le MDA fournit le niveau d'abstraction nécessaire permettant de supporter la complexité des applications et la prolifération des plateformes technologiques grâce à ces deux principaux artefacts : les modèles et leurs transformations. Il apporte un gain en productivité dans le développement des applications et la réduction du coût dans la migration entre plateformes technologiques grâce aux transformations de modèles (PIM vers PIM, PIM vers PSM, PSM vers PSM, PSM vers code, etc.).

L'introduction du concept de modèle par l'IDM a entrainé la définition d'un langage de représentation des modèles appelé langage de modélisation. Ce dernier va offrir des notations appropriées permettant de représenter les modèles à différents niveaux d'abstraction (cf. figure 1.2). La proposition initiale de l'OMG était d'utiliser le langage UML comme langage de modélisation des modèles PIM et la plupart des PSM. Ce choix se justifie par le fait qu'avec l'apparition du paradigme objet vers les années 1980, UML était devenu le standard incontournable des langages de modélisation objet. Il était déjà adopté et implanté dans les entreprises et supporté par plusieurs outils et un grand nombre d'AGL (Atelier de génie logiciel). Les nouveaux apports de sa version 2.0 le place ainsi au centre du MDA et fait de lui le langage de modélisation le plus important de cette approche. Par la suite, à cause de l'approche généraliste d'UML, l'OMG promeut l'utilisation d'autres langages de modélisation spécifiques aux domaines particuliers et à différents niveaux. Ces langages appelés langages dédiés ou DSL (*Domain Specific Langage*) sont définis grâce au principe de la méta-modélisation de l'IDM et doivent être conforme au standard MOF (*Meta Object Facility*) de l'OMG (OMG, 2006a) (cf. Section 3.4.1 pour plus de détails).

Cependant, malgré le niveau d'abstraction offert par les modèles, la pratique du MDA est restée seulement destinée aux ingénieurs logiciels et non aux experts métier car le processus commence avec les modèles PIM et non les modèles CIM (OMG, 2003a) (voir figure 1.2). Les modèles/langages métier décrits avec UML et les règles métier exprimées dans les langages basés sur XML (RuleML, RIF) par les ingénieurs logiciels ne sont pas facilement compréhensibles par les experts métier. L'OMG n'a émis aucune préconisation quant à l'élaboration des modèles/langages métier (CIM). Ces modèles/langages sont restés néanmoins beaucoup moins utilisés dans la pratique par la communauté MDA que les PIM et les PSM. Ainsi, le MDA va rester une approche de modélisation des SI orientés métier par les experts système.

Toutefois, le MDA demeure une approche séduisante mais reste un grand chantier car afin d'être plus efficace, il est indispensable que le processus de modélisation commence au point d'origine à savoir la spécification formelle (i.e. basée sur un méta-modèle ou une grammaire) des règles métier (CIM) et ceci par les experts métier en utilisant leur langage métier (naturel). C'est à ce niveau que les motivations d'une entreprise sont apparentes et leur modèles métier (vocabulaire métier, règles métier, processus métier) sont spécifiés (Hendryx, 2005). On pourra ensuite avoir des transformations (CIM vers PIM) qui contribueront à réduire le pont sémantique entre le métier et l'informatique et ainsi aligner les SI avec les objectifs de l'entreprise.

1.3 Les limites de l'existant

Nous avons vu que l'ARM et le MDA sont deux approches éprouvées et complémentaires dans la modélisation des SI qui tentent d'apporter des solutions aux questions posées plus haut dans la section 1.2. L'ARM vise la séparation de la logique métier (règles métier) de la logique système. La mise en œuvre des règles métier dans les SI est facilitée par l'intermédiaire des moteurs de règles (*rule engine*) qui se chargent de leur exécution à la demande des applications métier. Un meilleur alignement des SI avec la logique métier est ainsi permis, ce qui apporte un gain significatif en termes d'agilité (rapidité d'adaptation) face aux changements et de réutilisabilité de la logique métier. L'ARM peut être mise en œuvre grâce à une architecture MDA qui promeut la modélisation des modèles/langages de règles à différents niveaux d'abstractions (CIM, PIM et PSM) du cycle de vie des règles métier et ceci pour différentes audiences (experts métier, informaticiens et machines). Le MDA préconise le passage d'un niveau à un autre par des transformations.

Cependant, au delà de cette idée séduisante et prometteuse, il y a encore des difficultés qui entravent l'adoption de ces deux approches à une plus grande échelle. L'enjeu majeur reste la gestion de l'ambigüité du langage métier pour permettre l'expression formelle des règles métier par les experts métier au niveau de la couche CIM et leurs transformations vers des modèles PIM et ensuite vers des modèles PSM pour les moteurs de règles.

Plus encore, les avancées et la communauté grandissante de BRMS ajoutent de nouveaux besoins pour l'ARM, tels que la flexibilité et l'interopérabilité entre les BRMS. La flexibilité et l'interopérabilité sont devenues l'intérêt majeur des entreprises et cela pour améliorer leur productivité, leur fiabilité et leur rapidité d'adaptation aux changements. Comme la plupart de

ces BRMS sont commerciaux (JRules[4], Blaze Advisor[5] (Vincent et Isaac, 2005), Jess (Friedman-Hill, 2003) etc.), leurs langages de règles sont propriétaires. Il n'existe aucun système de transformation pour une paire d'entre eux afin de faciliter l'interopérabilité entre différents langages de règles.

Pour ces raisons, l'un des besoins le plus pressant et indispensable pour le succès de l'ARM reste non seulement la définition d'un langage intuitif aux experts métier pour l'expression des règles métier, mais aussi d'un standard pour faciliter l'échange des règles entre les différents langages/moteurs de règles. La standardisation est d'autant plus importante pour des règles sous-jacentes aux réglementations car elles pourront être partagées entre toutes les organisations concernées, favorisant l'interopérabilité sémantique de la conformité réglementaire et réduisant ainsi le risque de multiples interprétations des textes législatifs.

Les principales communautés de standardisation à savoir l'OMG et le *World Wide Web Consortium* (W3C)[6] ont longtemps concentré leurs efforts à fournir des spécifications des langages de règles métier afin d'obtenir l'interopérabilité des règles. Toutefois, la plupart de ces langages ne s'adressent pas aux experts métier car ils appartiennent aux couches PIM et PSM du MDA.

Par exemple, à la couche PIM du MDA, plusieurs tentatives de standardisation des langages de règles existent mais n'ont pas eu le succès attendu. Certains tentent d'adresser tous les types de règles et de moteurs qui existent à l'instar de RuleML de l'initiative RuleML[7], certains adressent juste une catégorie de règles. On peut entre autre citer : CommonRule (Grosof et Labrou, 2000) d'IBM, OCL (OMG, 2006b) et PRR (OMG, 2009) de l'OMG, RIF[8] du W3C. La plupart sont des langages à balisage (basés sur XML). Le but principal d'une approche de balisage est de fournir des moyens pour la réutilisation, la publication et l'échange de règles entre les différents systèmes, outils et ceci même à l'échelle d'internet. En fait, ils jouent également un rôle important dans la facilitation des interactions *business-to-*

[4] http://www-01.ibm.com/software/info/ilog

[5] http://www.fico.com/en/products/fico-blaze-advisor-business-rules-management-system

[6] W3C - http://www.w3.org

[7] http://ruleml.org/index.html

[8] http://www.w3.org/TR/rif-overview

customer (B2C) et business-to-business (B2B) sur internet. À la couche PSM, on a le standard JSR-94 qui permet l'échange de règles entre les moteurs de règles basés sur le langage de programmation Java.

Plus récemment, l'OMG a publié le standard *Semantics of Business Vocabulary and Business Rules* (SBVR) (OMG, 2008a) qui ne s'adresse qu'aux experts métier et se positionne à la couche CIM du MDA. SBVR ne standardise pas un langage de règles particulier ou n'est pas en soi un langage pour exprimer le vocabulaire et les règles métier, mais spécifie un méta-modèle (syntaxe abstraite indépendante de tout langage) pour définir le modèle (structure) sémantique du vocabulaire et des règles métier exprimés en langage naturel utilisé par les experts métier (OMG, 2008a). Avant SBVR, l'idée de méta-modèle sémantique avait été déjà explorée dans (El Abed, 2000) qui a proposé un méta-modèle sémantique pour l'interrogation multilingue des bases de données en langue naturelle. En d'autres termes, SBVR établit un vocabulaire pour parler du sens, du vocabulaire et des règles métier (Anderson et Spreeuwenberg, 2009). Une partie du méta-modèle appelée formulation sémantique (*Semantic Formulation*) permet de décrire la structure du sens (composition logique du sens) des règles et des définitions des concepts pour permettre une interprétation cohérente des concepts et des règles (Linehan, 2008). Bien que SBVR ne dise rien sur l'automatisation des règles métier car appartenant à la couche CIM du MDA, le fait qu'il soit basé sur la logique formelle (logique de prédicat de premier ordre) pourrait constituer le point de départ pour des transformations vers des modèles exécutables. À cet effet, le standard spécifie un format d'échange basé sur le standard XMI (*XML Metadata Interchange*) pour permettre l'échange du vocabulaire et des règles métier entre les différents outils.

Le principe de SBVR est intéressant et constitue ainsi le standard support pour l'ARM. Cependant, il est restreint juste à la sémantique en laissant de coté une caractéristique clé du langage naturel qui est la syntaxe. Pour diverses raisons, le standard n'a pas défini une spécification normative de la syntaxe à utiliser pour l'expression du vocabulaire et des règles métier (Ruth, 2009) par les experts métier. En plus, le méta-modèle SBVR est très complexe[9] (Linehan, 2008 ; Klener, 2009 ; Ruth, 2009) et les structures de sens (modèles sémantiques) des règles et des définitions des concepts décrites par le méta-modèle ne peuvent pas être directement utilisables par les experts métier (exprimée en XML) (Anderson et

[9] Décrit dans un document de 434 pages et est composé de 109 méta-classes.

Spreeuwenberg, 2009). Les modèles sémantiques sont à destination des outils et sont peu lisibles pour les experts métier. Ainsi, pour tirer profit de SBVR, les règles métier doivent tout d'abord être définies dans un langage métier avant d'être ensuite « mappées » vers des modèles sémantiques SBVR.

1.4 Contributions de la thèse

Cette thèse s'inscrit en droite ligne dans cette problématique et consiste à répondre à la question suivante :

Comment réduire la distance sémantique entre le langage naturel des experts métier et le langage formel des règles en passant par la sémantique SBVR ?

Cette question renvoie à la modélisation en langage naturel dont l'objectif principal est de rendre le langage naturel approprié pour la modélisation conceptuelle. Une solution prometteuse est l'utilisation des langages contrôlés (LCs) qui puisse servir comme interfaces frontales aux experts métier et des transformations automatiques vers des modèles sémantiques SBVR. Dans le domaine du traitement automatique des langues (TAL), les LCs qui sont des sous ensembles du langage naturel avec une restriction faite sur le lexique, la syntaxe et la sémantique ont beaucoup contribué à la réduction de l'ambigüité du langage naturel. L'avantage est qu'ils ont une syntaxe bien définie et une sémantique très précise. De ce fait, ils sont appliqués dans des applications variées du TAL comme la traduction automatique, la représentation des connaissances, etc. Pour cela, les recherches que nous menons dans cette thèse consistent en une montée en abstraction suivant l'approche de méta-modélisation du MDA pour proposer :

1) Un langage naturel contrôlé ou *Controlled Natural Langage* (CNL) ou tout simplement langage contrôlé que nous appellerions RuleCNL pour l'expression des règles métier par des experts métier.

2) Des transformations automatiques depuis un modèle de RuleCNL vers des modèles sémantiques SBVR. Étant donné que SBVR est entièrement intégré dans l'architecture MDA (cf. Chapitre 3) et fait partie des modèles métier (CIM), ces modèles sémantiques SBVR formels (basés sur la logique de prédicat) pourront être soumis aux transformations MDA pour aller vers d'autres langages formels de règles ou des modèles exécutables (modèles PIM et PSM du MDA). Mais cela n'est pas traité dans cette thèse.

3) Un outil support (*proof of concept*) qui fournit des interfaces conviviales avec tous les composants nécessaires pour assister un experts métier dans l'utilisation de RuleCNL.

La figure 1.3 donne une vue globale de RuleCNL dans une architecture MDA.

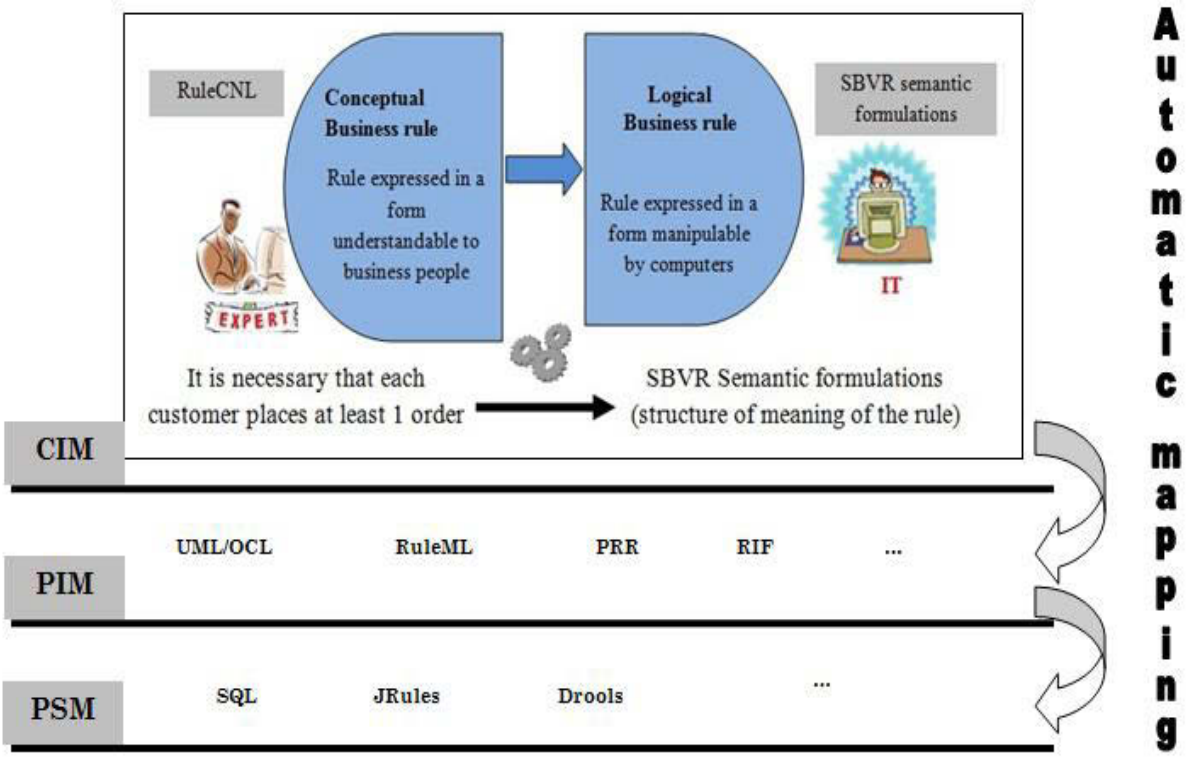


Figure 1.3 : Vue globale de RuleCNL dans une architecture MDA

Par soucis de lisibilité, la formulation sémantique SBVR de l'exemple de la figure 1.3 est représentée à la figure 1.4 ci-dessous. Nous l'avons représenté sous forme arborescente pour faciliter la lisibilité de la structure, mais elle peut également être représentée en XML.

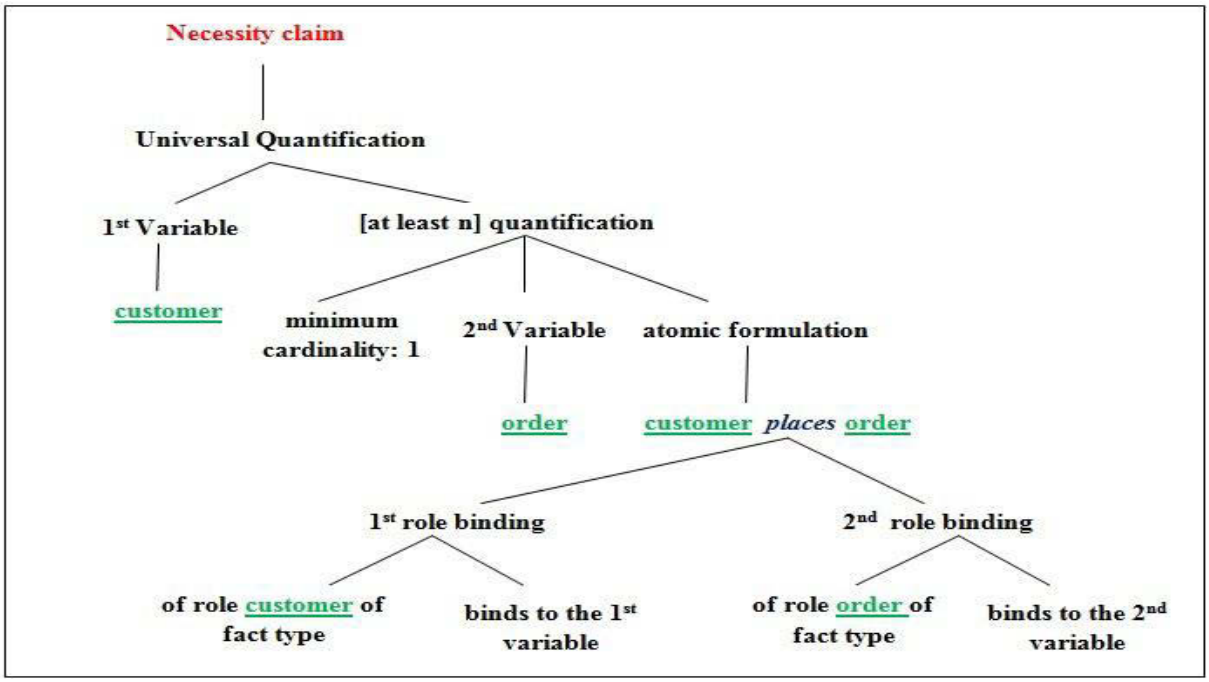


Figure 1.4. Formulation sémantique de la règle métier donné sur la figure 1.3

La structure arborescente de cette formulation sémantique se lit ainsi :

The rule is a proposition meant by a necessity formulation
. That necessity formulation embeds a universal quantification
.. The universal quantification introduces a first variable
... The first variable ranges over the concept "customer"
.. The universal quantification scopes over an at-least-n quantification
... The at-least-n quantification has the minimum cardinality 1
... The at-least-n quantification introduces a second variable.
.... The second variable ranges over the concept "order"
... The at-least-n quantification scopes over an atomic formulation
....The atomic formulation is based on the fact type "customer places order"
..... The atomic formulation has a first role binding.
.....The first role binding is of the role "customer" of the fact type.
.....The first role binding binds to the first variable.
..... The atomic formulation has a second role binding.
..... The second role binding is of the role "order" of the fact type.
.....The second role binding binds to the second variable.

1.4.1 Notre méthodologie

Notre méthodologie est basée sur un alignement (sémantique) de la spécification de la règle métier avec le vocabulaire métier. Ceci permet d'assurer la consistance et la traçabilité avec le domaine métier et ainsi de rendre RuleCNL indépendant du domaine métier.

Comme tout langage (naturel, formel, programmation, etc.), la définition de RuleCNL passe par une syntaxe et une sémantique. La syntaxe de RuleCNL est constituée d'une syntaxe abstraite définie par un méta-modèle et d'une syntaxe concrète textuelle définie par une grammaire. Le fait de dissocier la syntaxe abstraite de la syntaxe concrète est une caractéristique importante qui donne la possibilité d'avoir plusieurs syntaxes concrètes (ex. français et anglais) pour RuleCNL. La sémantique d'exécution de RuleCNL est une sémantique par traduction (Kahlaoui, 2011) définie par des transformations vers les modèles SBVR.

1.4.2 Notre approche de conception

Pour sa mise en œuvre de notre RuleCNL, nous adoptons une approche pluridisciplinaire. Nous combinons les techniques de l'IDM autour de ses deux principes clés : (1) la méta-modélisation (Atkinson et Kühne, 2002) (2) les transformations de modèles (Sendall et Kozaczynski, 2003) et du TAL autour de la spécification des grammaires.

La méta-modélisation est l'activité qui nous permet de définir le méta-modèle de RuleCNL qui est notre système à modéliser. Ce méta-modèle est la spécification de sa syntaxe abstraite à savoir le lexique des concepts du langage (vocabulaire), les propriétés de ces concepts, et les relations entre ces concepts. Ainsi, une phrase de RuleCNL (i.e. une règle métier) est représentée par un modèle qui est conforme à ce méta-modèle (cf. Section 3.4.1). Les transformations de modèles s'appuient sur ce méta-modèle de RuleCNL et celui de SBVR pour générer un modèle SBVR à partir d'une règle métier de RuleCNL. Puisque RuleCNL a une syntaxe concrète textuelle (sous-ensemble du langage naturel), nous spécifions une grammaire à partir du méta-modèle qui définie les règles de la syntaxe et sert à construire le parseur pour l'outil support de RuleCNL.

1.5 Organisation du document

Cette thèse couvre deux grands domaines de recherche importants qui sont l'ARM et les langages contrôlés. Après ce chapitre introductif, le reste de ce document est structuré en cinq chapitres.

Nous présentons au Chapitre 2 l'approche par règles métier dans la modélisation des systèmes d'information. Nous commençons en donnant une vue d'ensemble sur les règles métier. Ensuite, nous exposons les motivations derrière l'ARM et enfin nous détaillons sa mise en œuvre.

Dans le Chapitre 3, nous présentons l'ingénierie dirigée par les modèles autour des standards de l'OMG : le MDA et le SBVR. Après une vue macroscopique de l'approche MDA, nous présentons ses deux principes clés qui permettent la mise en œuvre de notre langage contrôlé RuleCNL ; il s'agit de la méta-modélisation et les transformations des modèles. Enfin, nous présentons en détail le méta-modèle SBVR qui est le standard phare de l'ARM et constitue le domaine sémantique de RuleCNL.

Le Chapitre 4 dresse un état de l'art profond sur les langages contrôlés qui sont de bons médiateurs entre les langages naturels des humains et les langages des machines. Nous présentons tout d'abord leurs origines et leurs définitions, puis leurs différents types avec les diverses applications associées. Nous terminons par un état de l'art des langages contrôlés pour les règles métier.

Dans le Chapitre 5, nous présentons notre langage contrôlé RuleCNL pour la spécification des règles métier qui constitue les principales contributions de cette thèse. Nous présentons tour à tour son vocabulaire, sa syntaxe, sa grammaire, sa sémantique d'exécution, son implémentation ainsi que son évaluation.

Le Chapitre 6 présente la conclusion générale, le bilan des contributions ainsi que les perspectives de la thèse.

À la fin de la thèse, trois annexes sont présentées. L'annexe I présente le manifeste pour les règles métier défini par le BRG. L'annexe II présente le modèle complet des règles métier et l'annexe III montre la chronologie de l'évolution des langages contrôlés pour l'anglais.

Chapitre 2

Approche par règles métier dans la modélisation des systèmes d'information

2.1 Introduction

2.1.1 Notion de règles

Les règles sont très importantes en général car elles existent partout. Chaque pays a des règles (ou encore des lois), chaque entreprise a des règles, chaque institution a des règles, chaque communauté a des règles, chaque ménage a des règles, chaque couple a des règles, chaque personne a des règles, etc. Par exemple, si une personne décide de se lever chaque jour à 7h afin d'être à son lieu de service à l'heure, c'est une règle. Si un étudiant doit passer son examen avec une moyenne supérieure ou égale à 10/20, c'est aussi une règle, etc. Les règles ont toujours tendance à réduire notre degré de liberté. S'il n'y avait pas de règles, on vivrait dans une liberté totale qui pourrait se transformer en anarchie. Si certaines règles sont données et n'ont pas tendance à réduire notre degré de liberté, elles pourront bien être utiles, mais ne pourront pas être considérées comme des règles en soi, mais comme des conseils (Ross, 2013). On peut trouver les règles sous plusieurs formes, allant des formulations simples aux formulations très complexes. Certaines peuvent être correctement documentées, certaines existent seulement dans la tête de ses parties prenantes. La mise en application ou le respect de certaines règles peut faire l'objet d'un renforcement, elle peut être seulement basée sur le bon sens ou sur la confiance de ses parties prenantes.

2.1.2 Notion de règles métier

Plus encore, dans le cas particulier des entreprises, les règles encore appelées règles métier[10] (*Business Rules*) sont d'autant plus importantes et sont au cœur de toutes les opérations, des transactions et à la base des prises de décisions dans l'entreprise. Les processus métier et les systèmes d'information sont régis par des règles métier. Elles encapsulent en leur sein la

[10] Dans la littérature, on peut l'appelle aussi règles de gestion ou règles d'affaires.

logique métier de l'entreprise. Cependant, le problème récurent dans les entreprises est que certaines de ces règles métier sont implicites et donc mal appliquées, d'autres sont documentées mais pas bien appliquées, et d'autres encore sont peut-être mal écrites et obscurément appliquées. Comme déjà mentionné au Chapitre 1, ceci est dû au fait que les approches traditionnelles de modélisation des systèmes d'information ne permettent pas une gestion propre et centralisée de ces règles métier. Elles sont très souvent dispersées dans différentes applications métier et à différents endroits de façon redondante et incohérente. Par exemple, une même règle métier peut être implémentée à plusieurs endroits différents. En plus, elles sont complètement inaccessibles aux experts métier et ne peuvent être modifiées que par des experts système. Ceci rend les systèmes d'information sclérosants et conduit à de nombreux inconvénients car les entreprises font face à plusieurs changements de nos jours : elles doivent relever le défi de s'adapter à la concurrence sur le marché, de vite prendre en compte les nouveaux produits et clients, de suivre l'évolution des préférences de ses clients, de se maintenir dans un environnement métier évolutif, etc. Tous ces défis exigent des changements rapides et rigoureux des opérations et des décisions de l'entreprise. Ces opérations et décisions sont à leur tour basées sur des règles métier, des contraintes et des politiques qui doivent aussi changer en conséquence. Pour des raisons citées plus haut, ces changements deviennent coûteux et nécessitent beaucoup de temps et d'effort de la part des experts système, puisque la logique métier est inaccessible aux experts métier et répercutée dans les applications. Les entreprises modernes et compétitives ont besoin d'avoir des systèmes d'information qui puissent s'adapter très rapidement et à faible coût aux changements dès que le besoin se fait ressentir afin de réagir rapidement sur de nouvelles opportunités métier. Les systèmes d'information ne doivent pas constituer un frein ou un obstacle à l'évolution, mais plutôt un catalyseur pour des solutions métier. Pour que cela soit possible, les experts métier doivent (Boyer et Mili, 2011) :

(1) connaître exactement les règles métier qu'ils utilisent, et s'ils les utilisent de façon cohérente.

(2) décrire les règles métier qui sont implémentées dans leur SI de telle manière que toutes les parties prenantes (métier, système, etc.) puissent comprendre, et ils doivent disposer d'un moyen de garantir de la traçabilité et la consistance entre la description de la règle et son implémentation réelle dans les systèmes automatisés.

(3) utiliser des outils métier agiles qui leur permettent de réagir à l'évolution des solutions métier au moment opportun

L'approche par règles métier que nous présentons dans ce chapitre aborde les trois points évoqués ci-dessus et essaye d'apporter des solutions. Pour cela, elle place les règles métier au centre du processus de modélisation afin de permettre la mise en œuvre des SI orientés métier et par le métier. Dans la suite, après avoir présenté une vue d'ensemble du composant fondamental de l'ARM qui est la règle métier, nous présentons les motivations derrière l'ARM ainsi que sa mise en œuvre.

2.2 Vue d'ensemble sur les règles métier

2.2.1 Origine des règles métier

Le concept de règle métier trouve ses origines principalement dans la communauté de l'intelligence artificielle (IA) où elles ont été largement utilisées pour l'ingénierie des connaissances depuis la fin des années 1960 (Bajec et Krisper, 2004). Dans les années 1970, les systèmes experts (SE) sont apparus comme une partie identifiable de l'IA et la logique a commencé à être utilisée comme un langage de programmation (Prolog). Traditionnellement, les systèmes experts sont des programmes qui peuvent remplacer les experts humains dans une certaine mesure afin de rendre leur connaissance sous forme de règles plus accessible. Ces règles sont généralement décrites à l'aide des langages déclaratifs et stockées dans une base ou un référentiel de règles. Un moteur d'inférence évalue à tout moment les conditions des règles afin de déterminer celles qui doivent être appelées. La recherche sur les systèmes experts a commencé dans les années 1960 et plusieurs systèmes experts ont vu le jour. Dendral (Buchanan et al. 1969) a été l'un des premiers systèmes experts, initié en 1965. Il a été conçu pour aider les chimistes à analyser les molécules organiques et a été une réussite. Encouragé par cette réussite ainsi que d'autres, les systèmes experts sont devenus très populaires dans les années 1970 et 1980. R1 (plus tard appelé XCON) (McDermott, 1981) est un autre exemple d'un système expert à succès. Il a été développé par le *Digital Equipment Corporation* (aujourd'hui connu sous le nom de Hewlett-Packard) et pouvait sélectionner automatiquement les configurations appropriées des composants des systèmes informatiques sur la base du bon de commande d'un client. EMYCIN (Shortliffe, 1976) fût également un autre système expert qui pouvait diagnostiquer des maladies infectieuses du sang et avec un

certain succès aussi. Cependant de nombreux autres systèmes n'ont pas réussi à avoir le même succès et seule une minorité d'entre eux a connu une utilisation très répandue (Grandon, 1995). Les systèmes experts étaient conçus pour fonctionner comme des systèmes fermés (en isolation) et ne pouvaient être associés à des systèmes opérationnels de l'entreprise. Ceci a constitué un énorme frein et le domaine des systèmes experts a progressivement disparu du paysage scientifique dans les années 1990. Toujours dans les années 1990, il a été proposé d'utiliser l'anglais contrôlé dans les systèmes experts afin de rendre l'acquisition et la maintenance des connaissances plus conviviale aux utilisateurs (Pulman, 1996). Des systèmes modernes de gestion des règles métier avec des architectures plus flexibles et robustes pouvant être intégrés en partie ou entièrement dans les systèmes d'information conventionnels ont par la suite vu le jour.

Les premières discussions sur la gestion des règles métier ont aussi émergé dans la communauté des bases de données vers la fin des années 1980 dans un journal appelé *Database Newslettter* (Graham, 2006). Comme résultat, on est passé à l'émergence des bases de données relationnelles actives vers le début des années 1990. Contrairement aux bases de données passives où les actions sur les données sont des appels explicites d'un programme d'application, les bases de données actives implémentent en leur sein certaines règles pouvant être déclenchées automatiquement afin d'assurer l'intégrité des données. Cependant, les premières implémentations des règles métier dans ces bases de données étaient très limitées. Les règles étaient implémentées comme des procédures stockées de façon procédurale ou sous forme déclarative avec des extensions propriétaires de SQL (*Structured Query Language*) (Graham, 2006). D'autres règles, notamment les contraintes d'intégrités étaient implémentées dans le système de base de données lui-même, mais la tâche la plus complexe était d'améliorer cette approche. Après quelques années de recherche, les bases de données actives ont intégré des déclencheurs (*triggers*) qui sont des règles de la forme SI/Alors qui font des vérifications sur des mises à jour des valeurs entrées dans la base de données.

Ainsi, la plupart des pionniers et auteurs des ouvrages sur les règles métier viennent soit de la communauté de l'IA, soit de la communauté des bases de données. Dans ces deux communautés, un regard différent est porté sur la terminologie de la règle métier. Dans les systèmes experts, les règles métier sont plutôt considérées comme des règles de la forme *Si condition Alors action* et dans la communauté des bases de données, les règles sont plus des contraintes sur les données ou sur les attributs des données (Graham, 2006).

Une étape importante vers le rapprochement des deux communautés était la création vers les années 1989 du *Business Rule Group* pour la constitution des fondements de bases des systèmes de gestion de règles métier ainsi que les principes de l'approche par règles métier. La plupart des membres du groupe sont des spécialistes des technologies de l'information et viennent de la communauté des bases de données. Le premier objectif du groupe était focalisé sur les règles métier qui pourraient être directement automatisables dans les SI. Il s'agit notamment des règles qui seraient définies formellement dans les exigences des besoins des systèmes d'information. Au fil du temps, le groupe s'est concentré sur les aspects métier des règles métier, plutôt que des règles automatisables. Cela a conduit à la naissance de certains standards de l'OMG comme : SBVR et le BMM (*Business Motivation Model*). Un certain nombre d'ouvrages importants sur les règles métier sont issus de ce groupe parmi lesquels (BRG, 2000), (Ross, 2003), (Von halle, 2002), (Date, 2000), (Morgan, 2002). Le BRG travaille sur les règles métier depuis la fin des années 1980 et a publié un Manifeste présentant les notions clés de l'approche par règles métier avec dix principes caractérisant les règles métier. Il est disponible en Annexe I.

2.2.2 Le modèle conceptuel des règles métier

Le BRG a présenté dans un rapport initialement appelé le *GUIDE Business Rules Project* (BRG, 2000) le premier modèle conceptuel de règle métier qui présente l'origine des concepts et structures d'un méta-modèle pour la description d'un formalisme de règles métier. La figure 2.1 montre la première partie du modèle conceptuel des règles métier tandis que le modèle complet est présenté en Annexe II.

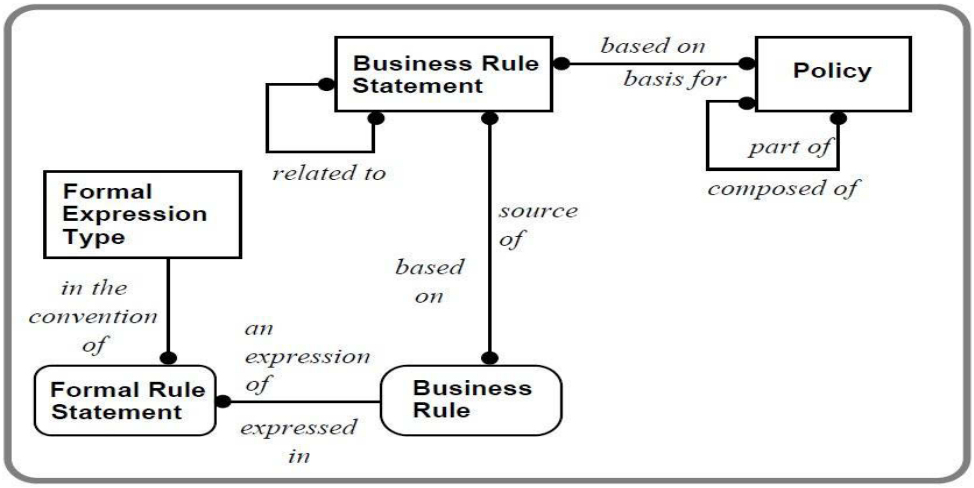


Figure 2.1: L'origine des règles métier (BRG, 2000)

Dans ce modèle, nous avons :

- «*Policy*» : qui est une déclaration qui définit la politique générale et les objectifs d'une entreprise. Chaque «p*olicy*» peut être composée de plusieurs «*policy*» plus détaillées, c'est-à-dire, une «p*olicy*» détaillée peut faire partie d'une ou plusieurs «p*olicy*» générales.

  Exemple: «*We only rent cars in legal, roadworthy condition to our customers*»

Un «*policy*» peut être à la base d'une ou plusieurs déclarations de règle métier «*Business Rule Statement*» et vice versa.

- «*Business Rule Statement*» : qui est une formulation déclarative de la structure ou des contraintes sur lesquelles se base le métier. Un «*Business Rule Statement*» peut être lié à un ou plusieurs autres «*Business Rule Statement*»

Exemple : Chacune des déclarations pourraient être des «*Business Rule Statement*»

"Cars should be checked on return from each rental, and on transfer between branches."

"If any lights are not working, the bulbs should be replaced. If tires are worn, they should be replaced."

"Under any of the following conditions the car should be scheduled for service or repair:

- *accumulated mileage since the last service is greater than 5000,*
- *the brakes are not satisfactory,*
- *the exhaust is noisy or emitting fumes,*
- *there is any damage to body work (apart from superficial dents and scratches), lights or glass,*
- *there are any significant fluid leaks."*

En fin de compte, un «*Business Rule Statement*» peut être à son tour source d'un ou de plusieurs «*Business Rule*» atomiques.

- «*Business Rule*» : comme un «*Business Rule Statement*», un «*Business Rule*» est une formulation qui définit ou contraint certains aspects du métier, mais à la différence du «*Business Rule Statement*», un «*Business Rule*» doit être atomique, c'est-à-dire qu'il ne peut être décomposé en d'autres «*Business Rule*» sans perte d'information. Chaque «*Business Rule*» peut être basé sur une ou plusieurs «*Business Rule Statement*»

Exemple : *"A car with accumulated mileage greater than 5000 since its last service must be scheduled for service."*

Une «*Business Rule*» peut être exprimée à l'aide d'un ou de plusieurs «*Formal Rule Statement*»,

- «*Formal Rule Statement*» : qui est l'expression d'une «*Business Rule*» dans un langage qui doit être dans la convention d'un «*Formal Expression Type*» spécifique, c'est-à-dire l'une des grammaires formelles pour représenter les règles métier telles que : ORM, RuleML, OCL, etc. Exemple : *If Car.miles-current-period > 5000 then invoke Schedule-service (Car.id) End if*

2.2.3 Définition d'une règle métier

An on-line store might not accept a next-day delivery order if the order is received after 3:00 p.m.

The bank will not lend money if the debt-over-income ratio exceeds 37%

A customer who places at least five orders is a golden customer

My health insurance does not reimburse medical expenses incurred abroad if the claim is presented more than 1 year after the expenses had been incurred, or if the claimant has spent more than 100 days abroad within the past year.

Ce sont ci-dessus quelques exemples de règles métier que nous pouvons trouver dans la pratique et dans différents domaines (e-commerce, banque, assurance, etc.).

Si nous décomposons le terme « règle métier », on aura règle du métier ou règle sous la juridiction d'un métier (domaine ou secteur d'activité). Bien que dans cette thèse, nous fassions plutôt référence aux entreprises en parlant de métier, le terme « métier » ne désigne pas seulement une entreprise privée commerciale. Il peut être considéré dans un sens très large et peut désigner une organisation ou une institution publique, une communauté ou n'importe quel autre domaine ou secteur d'activité (santé, assurance, banque, université, ministère, éducation, etc.).

Selon le dictionnaire électronique *Wordnet*[11], une *règle* est entre autre :

"A principle or condition that customarily governs behavior."

ou

[11] http://wordnetweb.princeton.edu/perl/webwn?s=rule

"A prescribed guide for conduct or action."

Une *règle du métier* donc signifie tout simplement que la prescription ou la condition définie par la règle est dictée par un domaine métier et ainsi, fait partie des besoins du métier (les exigences fonctionnelles du métier) par opposition à une prescription dictée par des choix technologiques (la solution au problème métier). Ainsi, les experts métier peuvent adopter, réviser, améliorer ou supprimer une règle métier comme ils entendent.

Il n'y a pas de définition standard généralement acceptée pour les règles métier (Hamza et Fayad, 2005). Toutefois, dans la littérature, de nombreuses définitions de règle métier ont émergées. Ces définitions ont évolué au fil du temps pour donner une définition partagée presque par tous. Nous pouvons citer entre autres quelques définitions données par les auteurs reconnus et cités internationalement comme les pionniers des règles métier :

(Morgan, 2002) définit de façon informelle une règle métier comme suit :

"*A compact statement about an aspect of the business [that] can be expressed in terms that can be directly related to the business, using simple, unambiguous language that's accessible to all interested parties: business owner, business analyst, technical architect, and so on."*

(Ross, 2003) définit une règle métier comme :

"A directive intended to influence or guide business behavior."

(Von Halle, 2001) voudrait que l'"on pense de la règle métier comme étant :

"The set of conditions that govern a business event so that it occurs in a way that is acceptable to the business."

(OMG, 2008) définit une règle comme :

"Proposition that is a claim of obligation or of necessity" et une règle métier comme une règle sous la juridiction du métier.

(Graham, 2006) définit une règle métier comme suit :

"A business rule is a compact, atomic, well-formed, declarative statement about an aspect of a business that can be expressed in terms that can be directly related to the business and its

collaborators, using simple unambiguous language that is accessible to all interested parties: business owner, business analyst, technical architect, customer, and so on. This simple language may include domain-specific jargon."

Le *BRG* a publié un certain nombre de documents traitant des règles métier et de la façon dont elles peuvent être représentées. La définition de la règle métier préconisée par le groupe et qui provient des définitions précédemment citées est la plus largement adoptée dans la littérature. Elle considère la règle métier selon deux points de vue : un point de vue métier et un point de vue de système d'information.

D'un point de vue métier (BRG, 2008) :

". . . *A business rule is guidance that there is an obligation concerning conduct, action, practice or procedure within a particular activity or sphere.*
Two important characteristics of a business rule:
(1) There ought to be an explicit motivation for it, and
(2) It should have an enforcement regime stating what the consequences would be if the rule were broken"

D'un point de vue SI (BRG, 2008) :

". . . *A business rule is a statement that defines or constrains some aspect of the business. It is intended to assert business structure, or to control or influence the behavior of the business*".

Cette distinction entre les deux points de vue considérés par le BRG est nécessaire compte tenu du fait que tout processus métier implique à la fois des acteurs humains, et des systèmes d'information, et les règles métier guident les deux. Dans cette thèse, puisque nous proposons un langage contrôlé pour la spécification des règles métier dans une perspective de modélisation des systèmes d'information, nous considérons la définition de la règle métier d'un point de vue système d'information. Toutefois, la règle métier exprimée doit pouvoir être à la fois compréhensible par les experts métier et par l'ordinateur. Une règle métier compréhensible par un expert métier signifie qu'elle doit être exprimée en utilisant le vocabulaire ou une ontologie (Roche, 2003a ; 2003b) et la sémantique du domaine métier. Une règle métier compréhensible par l'ordinateur signifie qu'elle doit avoir une syntaxe et une sémantique bien définies. Sa syntaxe doit être basée sur une grammaire formelle/méta-modèle afin d'être interprétée et/ou transformée automatiquement. Il s'agit donc d'un réel

compromis entre le langage naturel du métier et les langages formels exécutables. Ainsi, nous définissons la règle métier dans cette thèse comme suit :

*"A business rule is a **formal** statement that defines or constrains some aspect of the business. It is intended to assert business structure, or to control or influence the behavior of the business."*

Étant défini ainsi, une règle métier décrit les données capturées par des systèmes d'information à partir des entités du monde réel impliquées dans le processus métier telles qu'un client, un produit, ou une transaction. Elle exprime aussi des contraintes spécifiques sur la création, la mise à jour et la suppression de ces données dans les SI. Une caractéristique importante qui découle de cette définition est que la règle métier concerne à la fois la structure et le comportement du domaine métier. Cette caractéristique est très bien mise en perspective dans la définition de l'OMG qui sépare les règles métier en deux catégories : (1) Les règles métier structurelles ou de définition (*structural or definitional rules*) et (2) Les règles métier opérationnelles ou de comportement (*operative or behavioral rules*).

Après avoir défini ce que c'est qu'une règle métier, il reste encore beaucoup de chose à dire à son sujet, telles que la classification des règles métier, les qualités d'une bonne règle métier, etc.

2.2.4 Classification des règles métier

Afin de gérer l'ensemble des règles métier au sein d'une entreprise, les règles métier sont généralement classées suivant plusieurs catégories car il en existe plusieurs types. Elles peuvent être classées suivant plusieurs perspectives (métier ou système) et critères (automatisable ou non). De la même façon qu'il existe plusieurs définitions des règles métier dans la littérature, il existe également plusieurs classifications des règles métier possibles. Bien qu'il existe bon nombre de classifications suivant des critères différents, nous commencerons par la classification du BRG la plus citée dans la littérature qui par la suite a été étendue par d'autres auteurs.

2.2.4.1 Classification du BRG

Les premiers travaux sur les règles métier ont été faits par le BRG. Ainsi, la première classification des règles métier est celle du BRG publiée dans (BRG, 2000) qui considère une RM comme décrit sur la figure 2.2

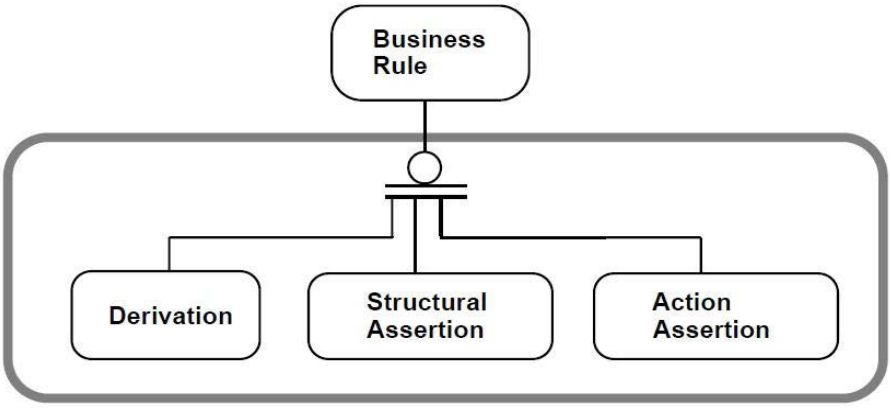


Figure 2.2 : Business rules Type (BRG, 2000)

❖ ***Structural assertion*** : Ce sont des règles métier qui définissent les concepts (termes métier) ainsi que les relations (faits) entre eux. Il s'agit des règles qui désignent la structure d'une entreprise et sont souvent implicites dans les modèles de données ou les diagrammes d'entités / classes. La définition d'un terme métier est en soi une règle métier qui décrit la manière dont les gens comprennent les choses. Les faits sont exprimés par des phrases dans le langage naturel, ou par des relations, associations ou autres formalismes dans des modèles graphiques.

❖ ***Action assertion*** : Ce sont des contraintes d'intégrités (règles d'intégrités) ou des conditions qui limitent ou contrôlent les actions dans une entreprise. Les règles d'intégrités sont des assertions qui doivent être satisfaites. Toute organisation contraint d'une manière ou d'une autre certains comportements. Dans les SI, ces contraintes se traduisent par la possibilité ou non de modifier des données ou de lancer certaines actions. Très souvent empêcher la mise à jour d'une donnée revient à empêcher la réalisation d'une action liée à cette donnée.

❖ **Derivation :** Il s'agit de règles métier complexes s'appuyant sur des faits dérivés d'autres faits par un mécanisme d'inférence ou par un calcul mathématique. Un fait dérivé

est traité ensuite comme un fait élémentaire car la dérivation n'est pas visible lorsque la règle métier référence le fait directement.

Cette classification est considérée comme le modèle de base ou de facto (Joubert, 2013) et a été étendue par plusieurs autres auteurs. Suivant le niveau ou le point de vue où l'on se situe, ces types de règles peuvent se décomposer en d'autres sous types de règles plus spécifiques.

2.2.4.2 Classification de l'OMG

En Janvier 2008, l'OMG a dans le standard SBVR classifié les règles métier suivant deux groupes et ceci d'un point de vue métier comme indiqué sur la figure 2.3 (OMG, 2008a). La spécification SBVR est décrite plus en détails au chapitre suivant.

(1) Les règles métier structurelles ou de définition (*structural or definitional rules*)
(2) Les règles métier opérationnelles ou de comportement (*operative or behavioral rules*).

Il faut noter que la classification de l'OMG se base essentiellement sur celle du BRG en s'appuyant tout simplement sur une vue métier. En marge de cette classification, l'OMG précise que dans le processus d'identification des règles métier dans une entreprise, il faut garder à l'esprit que les règles métier contrairement aux politiques métier (*Business Policies)* sont actionnables (ou praticables), non ambigües, et découlent de la politique métier. Dire qu'une règle métier est actionnable veut dire qu'un expert métier connaissant très bien la règle métier peut observer dans une situation particulière si l'entreprise est conforme ou non à la règle en question. Par contre, la politique métier (*Business Policy*) trace une ligne directrice qui doit être supportée par une ou plusieurs règles métier. Les règles opérationnelles contrairement aux règles structurelles peuvent être violées. Ainsi, elles peuvent avoir un régime de renforcement pour permettre leur mise en application. Une règle tend toujours à réduire le degré de liberté, si tel n'est pas le cas, la règle est définie comme étant un conseil (*advice*).

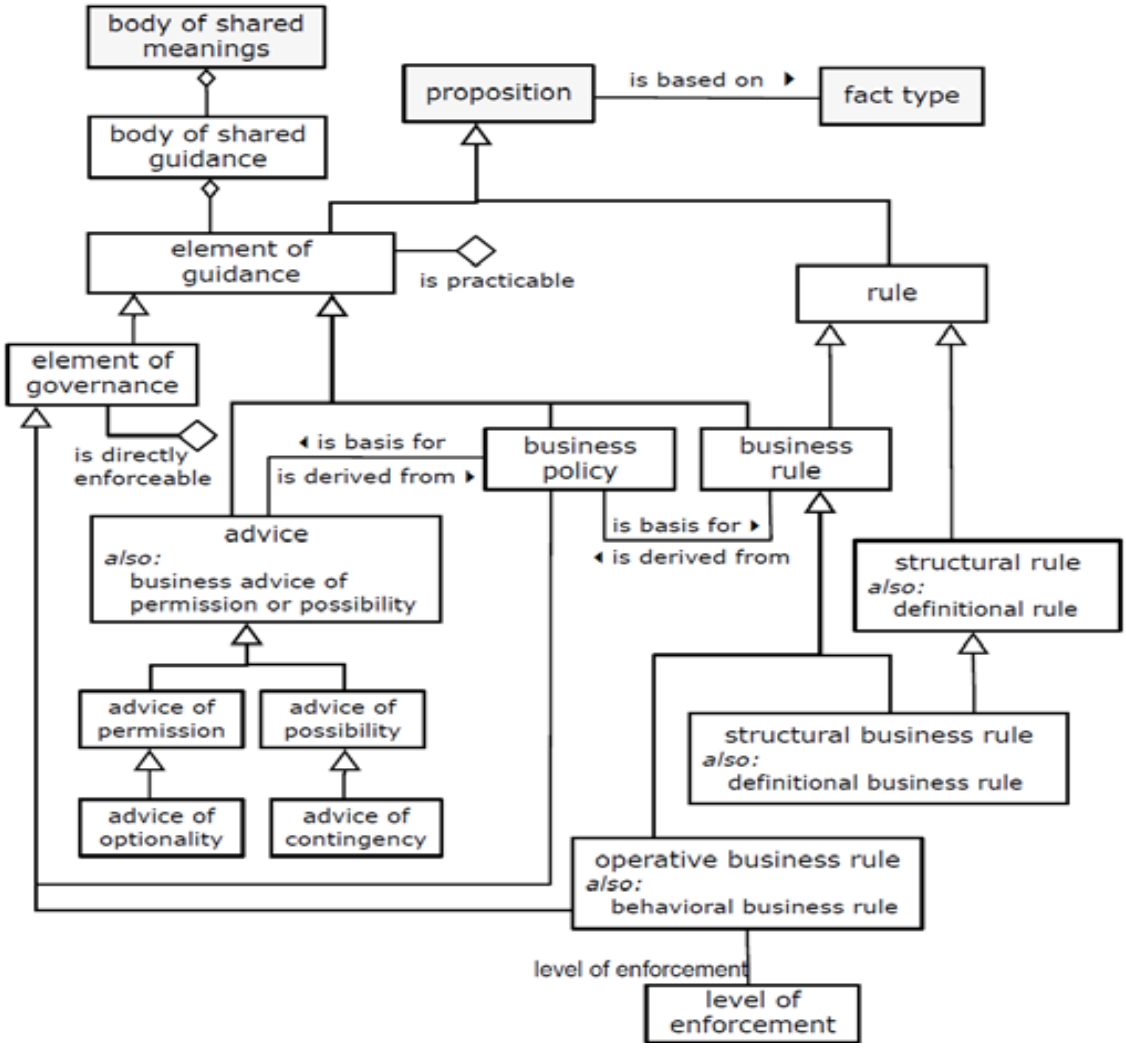


Figure 2.3 : Classification de l'OMG (extrait du document de spécification SBVR (p. 158))

2.2.4.3 Notre classification de règles métier

Les règles opérationnelles ont des répercussions fondamentales dans plusieurs secteurs de l'entreprise : Processus métier, SI, etc. Les opérations métier de base impliquent généralement un nombre important de règles opérationnelles. Elles sont à la base des prises de décisions et implémentent la logique décisionnelle de l'entreprise. Toutefois, l'OMG la définit d'un point de vue purement métier sans rien dire sur l'aspect automatisation. Il faut ici noter que le caractère actionnable ou praticable de la règle métier indiqué par l'OMG ne veut pas absolument dire automatisable. Ainsi dans le contexte des systèmes d'information où ces règles pourront être implémentées, nous avons besoin de subdiviser ces règles afin d'étudier leur caractère automatisable. Dans cet ordre d'idée, nous avons étendu le schéma de classification de l'OMG en plusieurs autres sous types de règles opérationnelles. Cette

extension (cf. figure 2.4) est un peu plus exhaustive et peut être pertinente pour une entreprise dans la mise en place d'une approche par règle métier dans son SI.

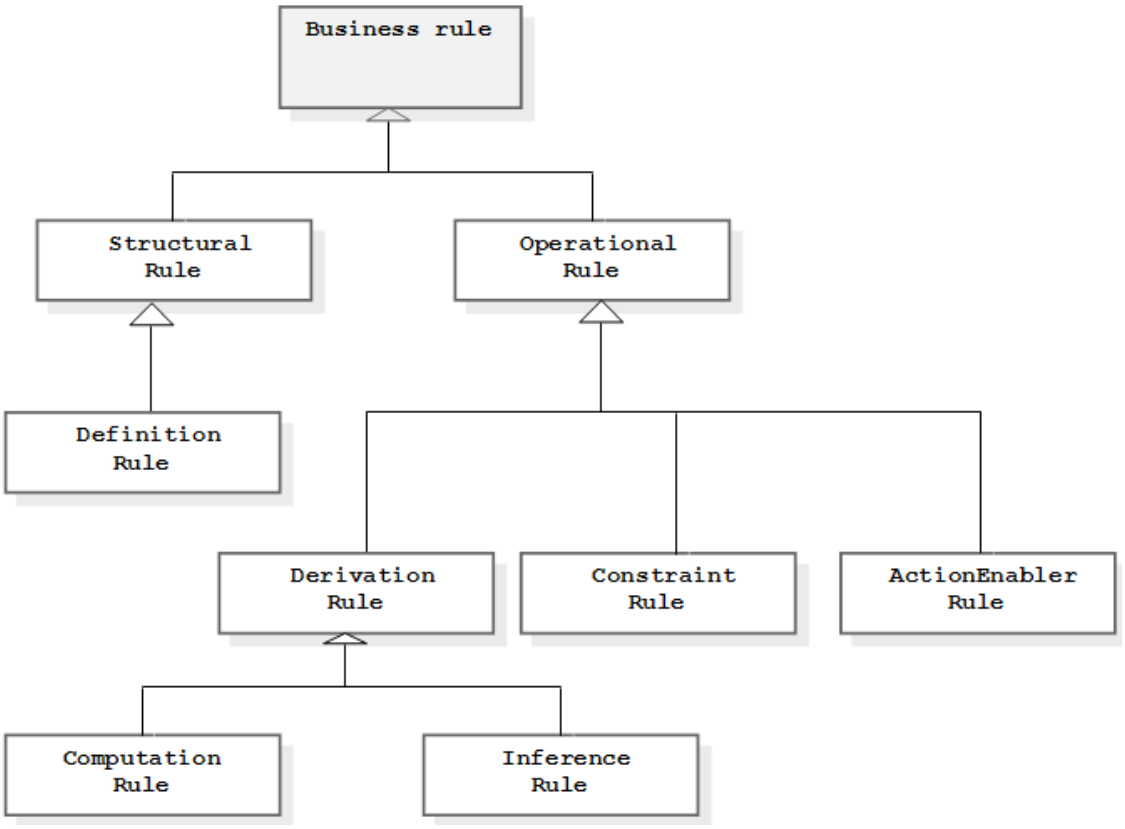


Figure 2.4 : Notre classification de règles métier

Ce schéma représente les différents types de règles qui sont les plus pertinentes pour une entreprise. Il a pour base les règles structurelles (*Structural rules*) et opérationnelles (*Operational rule*) de l'OMG. Les règles structurelles définissent les entités métier utilisées par le domaine métier dans l'expression des règles métier et les relations (faits) entre ces termes. Ceux-ci comprennent le vocabulaire utilisé dans la règle de création. À titre d'exemple, nous pouvons citer une règle comme :

An insurance policy includes a set of coverage. The policy is effective at a given date and needs to be renewed every six months.

Transformer cette déclaration implique la définition d'une structure dans le domaine de l'assurance, où une entité de la police d'assurance a une date de prise d'effet, une date d'expiration, et une liste de couvertures.

Dans cette classification, bien que nous parlions du caractère automatisable des règles métier dans la modélisation des SI, notre objectif dans cette thèse porte plus sur la représentation / spécification formelle des règles métier et ceci par les experts métier dans une approche par règles métier. Ainsi, notre vision sur les différents types de règles est plutôt focalisée sur la grammaire qui pourrait servir à définir leur syntaxe. Toutefois, étant donné que notre approche est basée sur l'architecture MDA (cf. section 1.2.2 du Chapitre 1 pour une description sommaire et le Chapitre 3 pour une description détaillée) qui place les modèles/langages de règles suivant trois niveaux d'abstractions, et que nous nous situons au niveau de la couche métier (CIM), cette classification des règles en amont facilitera le cycle de conception (couche PIM), qui est le point d'entrée pour une meilleure implémentation (couche PSM). Par exemple, les règles d'inférences (*Inference rules*) et les règles d'actions (*ActionEnabler rules*) sont des bonnes candidates pour les moteurs de règles. Les règles de calcul (*Computation rules*) pourraient être implémentées dans le code, sauf si elles sont sujettes à des changements fréquents dans les critères de l'applicabilité ou elles sont liées à d'autres règles métier. La classification permet également d'évaluer la complexité des règles et la charge de travail nécessaire pour son implémentation (Boyer et Mili, 2001). Il est aussi important de noter que ce n'est pas toutes les règles qui peuvent être implémentées et déployées dans une application métier. Certaines règles peuvent finir dans un manuel de procédure remis aux utilisateurs métier pour leur inculquer de bonnes pratiques métier tel que définie par la politique de l'entreprise et ne seront pas implémentées dans un composant logiciel. Cela inclut les règles utilisées par les décideurs humains qui nécessitent un jugement humain, ou qui nécessitent des données qui soit ne sont pas enregistrées par voie électronique, ou des données qui sont enregistrées mais ne sont pas utilisables.

Le tableau 2.1 ci-dessous présente une vue simplifiée de la grammaire pour la spécification des différents types de règles de la figure 2.4 ci-dessus. Nous présentons en détails la grammaire formelle de notre langage contrôlé RuleCNL au Chapitre 5.

Type de règles	Structure de représentation
Constraints	règles qui imposent des contraintes devant être vérifiées « term » MUST HAVE «at least, at most, exactly n of» «term»; «term» MUST BE IN LIST «a,b,c»; it is [not] necessary that «fact» it is [not] obligatory that «fact»
Definition	règles qui définissent les termes et les faits «term» SHOULD HAVE «at least, at most, exactly n of» «term»; «term» IS A «fact» it is [not] possible that « fact » It is necessary that « fact»
ActionEnabler	règles qui teste une condition et si condition vraie, elle lance une action, un événement, un processus, une acticité, etc. IF «condition» THEN action
Computations	règles qui dérivent de nouvelles informations à partir des informations existantes sur la base d'un calcul mathématique «term» IS COMPUTED AS «formula»
Inferences	règles qui dérivent de nouvelles informations à partir des informations existantes. Le résultat est à nouveau considéré comme un fait pour déduire d'autres informations IF «term» «operator» «term» THEN «term» «operator» «term»

Tableau 2.1 : Représentation des règles

Il est possible de poursuivre la décomposition de ces règles. Par exemple, les règles de transformations dans les outils ETL (*Extract Transform Load*) sont souvent considérées séparément des autres règles métier ; bien que dans leur représentation, ce sont essentiellement des règles d'inférences et les règles de calcul (Boyer et Mili, 2001). Les règles de transformation de données, bien qu'étant important pour l'entreprise, sont sujettes aux effets de bord et ne reflètent pas la logique métier de base. Pour l'implémentation, la décision d'utiliser un moteur de règles pour les règles de transformation de données dépend du fait que ces règles sont statiques, dynamiques, ou axées sur le métier. Certaines implémentations

utilisent un moteur de règles pour mettre en œuvre facilement les règles de transformations entre deux modèles de données au lieu d'utiliser un langage de script complexe lorsque les transformations doivent être maintenues par les experts métier. Ainsi, dans la mise en place d'une ARM, il est aussi important d'analyser la volatilité de la règle, c'est-a-dire la fréquence avec laquelle la règle doit changer. Ainsi, il est aussi important d'évaluer les facteurs qui déclenchent les changements de règles et comment de nouvelles règles sont définies pour un point donné de décision.

2.2.5 Qualité d'une bonne règle métier

Presque chaque déclaration ou formulation concernant une entreprise peut être considérée comme une règle métier. Un challenge auquel les entreprises sont confrontées avec l'ARM, est de trouver et recenser de bonnes règles métier. Le BRG a publié plusieurs exigences pour les règles métier (BRG, 2003), qui conduisent à de bonnes règles métier. Complétée par quelques exigences supplémentaires d'autres auteurs (Morgan, 2001), (Steinke et Nickolette, 2003) on peut citer entre autres qu'une règle métier est considérée comme bonne si elle possède les caractéristiques suivantes :

- **Atomique**

Elle ne peut pas être subdivisée. Une tentative pour décomposer une règle dans une version plus simple conduirait à une perte de sens. L'atomicité est très importante et recommandée pour permettre la compréhensibilité, la facilité de maintenance et l'efficacité de l'exécution. Par exemple, si nous considérons la règle :

The insurance does not reimburse medical expenses incurred abroad if the claim is presented more than one year after the expenses had been incurred, or if the claimant has spent more than 182 days abroad within the past year.

Cette règle n'est pas atomique et peux être décomposée en deux règles comme suit :

(1) If the date of creation of the claim is more than one year after the date of treatment of the medical expense then reject the medical expense.

(2) If the claimant spends more than 182 days abroad within the past year then reject the claim.

- **Exprimée en langage métier**

Elle doit seulement utiliser le vocabulaire et la terminologie du métier.

- **Consistent**

Une règle métier ne doit pas contredire l'autre. Ainsi, il est important ici de bien comprendre les dépendances entre les règles. Une règle R1 dépend d'une règle R2 si l'exécution de R2 résulte en une situation où R1 est sollicitée (ou doit être exécutée). Un exemple simple est une règle R2 qui crée de nouvelles données ou modifie les données existantes qui doivent être testées par R1. La compréhension des dépendances entre les règles aide à déterminer l'ordre d'exécution des règles. L'ordre d'exécution est utile dans l'analyse des règles pour détecter les dépendances indésirables. Pour l'implémentation, l'ordre d'exécution est utile et sert à comprendre à quoi les résultats vont ressembler: certains moteurs de règles déterminent cette séquence automatiquement et à la volée (chaînage). Si les règles métier sont implémentées de façon procédurale, l'ordre d'exécution est nécessaire pour leur application. Certaines dépendances indésirables incluent des dépendances circulaires menant à des boucles infinies.

- **Précise** : le sens de la règle doit être clair.

- **Déclarative**

Les règles doivent être exprimées d'une manière qui décrit les buts à atteindre plutôt que les séquences d'actions à faire pour atteindre ces buts. Une règle peut exprimer une condition « en cas de A, alors B », mais en aucun cas elle ne doit décrire les étapes à réaliser pour achever la transition (ou l'empêcher). Elle doit exprimer le but de la logique métier et pas comment l'obtenir.

- **Non ambiguë :** elle doit avoir une seule interprétation évidente.

- **Structurées :** même si la finalité des règles est d'être facile à lire et à comprendre, réduire les options pour les auteurs de règles permettra de faciliter l'automatisation. Ainsi, il faut qu'elles soient structurées.

- **La propriété du métier :** les experts métier doivent être capable de maintenir eux même les règles métier.

2.2.6 Identification des règles métier

Les entreprises fonctionnent avec les règles métier depuis de nombreuses années, mais ces règles ne sont pas explicitement gérées de façon autonome et centralisée. La procédure d'identification des règles métier est souvent itérative et heuristique. Au début, les règles ne sont que des formulations de politiques générales. Ces formulations sont parfois claires, parfois ambiguës, et souvent complexes (Von Halle, 2001). Les règles métier peuvent provenir de l'intérieur de l'entreprise (politique interne de l'entreprise, stratégie, vision, but, etc.) ou de l'extérieur d'une entreprise (lois, standards, gouvernement, faits intemporels naturels, etc.) (Herbst, 1996). Elles peuvent être basées sur des connaissances explicites (connaissances formalisées sous la forme de principes, de procédures, des faits, des schémas, des règles et des formules) ou tacites (savoir qui est difficile à voir et à exprimer).

Les règles métier sont du ressort des experts métier du domaine. Les règles métier peuvent provenir des déclarations de la politique de l'entreprise, d'un système d'information existant ou tout simplement de personnes travaillant dans l'entreprise et qui ont su acquérir au fil des années une expérience considérable. Cependant le plus difficile est de les recenser et de les structurer en une approche de management plus effective (Diouf, 2007)**.** Très souvent, les règles métier sont juste considérées comme des exigences métier dans les SI, mais elles sont plus larges et spécifient aussi comment un processus métier est exécuté, comment les décisions sont prises, et comment les connaissances du domaine sont structurées.

2.3 Motivations pour l'approche par règles métier

Les systèmes d'information jouent un rôle crucial dans les entreprises car ils automatisent les solutions (processus, applications, etc.) métier. Pendant plusieurs décennies, la recherche autour des méthodologies de modélisation des SI n'a pas cessé de croître. L'objectif récurrent était entre autre d'améliorer la qualité, la fiabilité des systèmes produits et d'augmenter la productivité en minimisant les coûts et les délais de développement afin de faire face à la complexité sans cesse grandissante des SI (distribués et de grande taille, etc.), De nos jours, en plus de cette complexité, les entreprises ont besoin des SI de plus en plus flexibles.
La flexibilité constitue aujourd'hui une préoccupation majeure des entreprises qui cherchent plus d'agilité et de réactivité. La flexibilité désigne la capacité d'une l'entreprise à agir et l'aptitude à répondre aux changements d'une manière dynamique et efficace (McCoy et

Plummer, 2006). La question de flexibilité de l'entreprise impacte nécessairement le domaine des systèmes d'information. Le système d'information, longtemps considéré comme un outil d'intendance au service de l'entreprise, est aujourd'hui au cœur de son fonctionnement, et sa flexibilité en conditionne les performances. Fort de ce constat, la flexibilité de l'entreprise se projette directement sur son système d'information qui doit s'aligner avec de nouvelles exigences et évolution métier. L'entreprise doit développer des capacités à agir et être extrêmement réactif afin d'aider à mettre sur le marché de nouvelles offres ou encore à répondre rapidement aux changements du contexte commercial. L'enjeu consiste donc à rendre les SI le plus flexible possible aux évolutions métier et de faciliter les prises de décisions par les experts métier. Ceci n'est guère possible que par une séparation nette entre la logique métier et la logique système. Les règles métier comme nous avons souligné à l'introduction constituent l'essence pour rendre ces SI flexibles et sont à la base de la logique métier. Les systèmes d'information implémentent généralement une large quantité de règles métier. Par exemple (Fu et al. 2004) montre que 627 règles métier sont appliquées dans une application COBOL de 12 000 lignes et 809 dans une application COBOL de 30 000 lignes. Un système d'information typique a une architecture à trois tiers comportant trois composants : une interface (généralement GUI), la logique applicative et une base de données. Les règles métier peuvent être implémentées dans différents endroits de ces trois composants. Cela peut causer divers problèmes avec la maintenance des règles métier. Ainsi, l'introduction d'une nouvelle couche ou composant pour la gestion des règles métier de façon indépendante constitue les motivations réelles de l'approche par règle métier dans la modélisation des systèmes d'information.

2.3.1 Approche traditionnelle de modélisation des systèmes d'information

Dans une approche traditionnelle (classique) de développement d'un SI, les experts métier expriment leurs besoins fonctionnels (règles métier) sous forme d'un cahier de charges. Ce dernier est remis aux ingénieurs logiciel (experts système) qui se chargent de concevoir l'application, l'implémenter, la tester, la valider, etc. Les experts métier qui sont à l'origine du système n'interviennent qu'en début du processus pour la spécification du cahier des charges de l'application et à la fin pour la validation de l'application et ne participent en aucun cas à la création du système comme indiqué sur la figure 2.5.

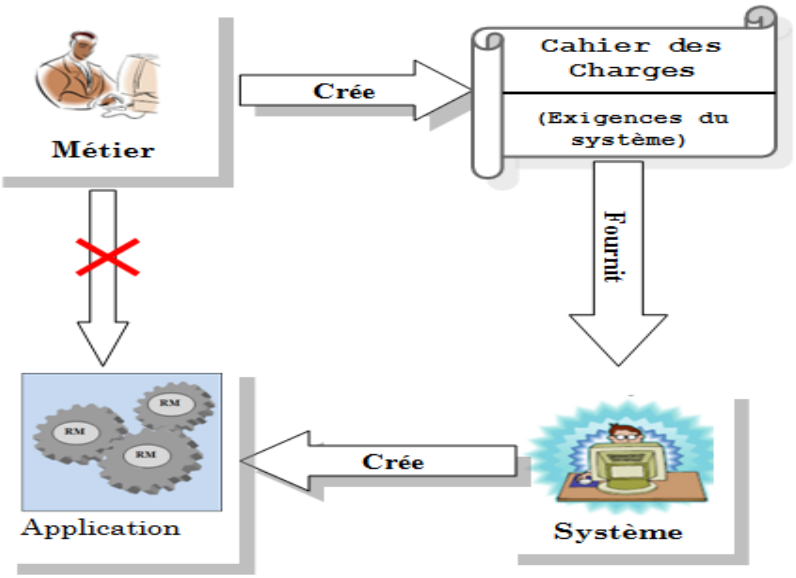


Figure 2.5 : Approche traditionnelle de modélisation des SI (Diouf, 2007)

Ils ne disposent pas de méthodes, ni d'outils pour faire évoluer le plus petit aspect du système sans l'aide des experts système. Une fois que l'application est déployée, si la politique de l'entreprise change, c'est-à-dire le cahier des charges ayant servi à réaliser le système, il faut faire intervenir de nouveau les experts système qui réétudient les besoins fonctionnels, re-analysent, re-conceptualisent, ré-implémentent, re-testent, revalident et re-déploient. Dans une telle approche, il est visible que l'expert métier, qui est au centre du métier, ne joue pas pleinement son rôle, ce qui rend la maintenance et l'évolution complexes rendant ainsi les SI moins flexibles. L'évolution est due aux changements importants et souvent rapides qui interviennent dans l'environnement métier (logique métier). Cependant, l'expert métier ne peut pas lui-même faire des changements sans intervention des experts systèmes. Dans un tel processus, l'échange entre les experts métier et système est limité car il n'a lieu qu'en début et en fin de processus. Très souvent, pendant le processus, les experts système s'attendent à ce que les experts métier comprennent les langages formels qu'ils utilisent afin d'y contribuer, ce qui n'est toujours pas possible car ces derniers n'ont pas (ou ne veulent pas avoir) de connaissances techniques sur les détails d'implémentations. Les conséquences peuvent ainsi être graves conduisant à des défaillances logicielles énormes ou des systèmes difficilement maintenables comme souligné au Chapitre 1. Selon (Standish, 1995), environ 66% de grands projets américains ont échoué à cause du manque d'implication des experts métier dans le processus de développement et des exigences des besoins mal spécifiées.

Il faut noter que de nos jours dans l'approche traditionnelle, beaucoup de progrès ont été accomplis quant à la rapidité (méthodes agiles) et la productivité des SI grâce à la montée en abstraction. L'arrivée de l'IDM a été la suite d'une longue évolution en génie logiciel et a considérablement contribué à la maitrise de la complexité des applications grâce au modèle qui est devenu le concept unificateur. Comme nous l'avons décrit (cf. Section 3.2 du Chapitre 3), on a assisté au cours des décennies à une évolution des paradigmes et techniques de développement. Le paradigme procédural a ainsi laissé la place au paradigme orienté objet qui, à son tour, a laissé sa place au paradigme orienté composant qui s'est enfin tourné vers l'IDM qui s'est fortement amplifié avec le standard MDA de l'OMG. Toutefois, il faut noter que malgré le niveau d'abstraction élevé offert par les modèles, le MDA est une approche technique de développement et reste cependant destiné aux experts système. Il leur permet de se soustraire aux détails techniques de l'implémentation en se focalisant dans un premier temps sur un niveau abstrait de modélisation et ceci indépendamment de la plateforme technologique, pour ensuite automatiser totalement ou partiellement la génération des détails techniques (code ou transformation de modèles). Ainsi, l'évolution et la maintenance restent coûteuses et réservées aux experts systèmes. La logique métier qui est à la base des décisions se trouve toujours dispersée dans la logique applicative gérée par les experts système.

La maitrise de cette évolution passe par une séparation nette de la logique métier (règles métier) des infrastructures applicatives afin de permettre la gestion des règles métier de façon centralisée au niveau métier. En plus, l'expert métier connaît mieux que quiconque les règles qui régissent l'entreprise et donc, du début de la spécification jusqu'aux tests en passant par l'implémentation, son rôle demeure indispensable. Les experts métier doivent pouvoir, eux-mêmes, créer et faire évoluer la logique métier pendant que les experts techniques ne s'occupent que des aspects techniques (systèmes) (Ross, 1997). Ainsi, il est nécessaire qu'en plus de la montée en abstraction offert par les modèles, l'approche de modélisation sépare très clairement la logique métier de la logique système.

2.3.2 Le principe de l'approche par règles métier

L'ARM est une méthodologie moderne et puissante qui répond à ces besoins et vise à améliorer de façon qualitative et quantitative les propriétés des approches traditionnelles (Butleris et Kapocius, 2002) en vue de mettre en œuvre des SI orientés métier et par le métier. Elle est complémentaire à l'IDM/MDA et introduit la notion de règles métier comme artefact central afin de surmonter les difficultés présentées par les approches traditionnelles. Elle est

née de la vision que les professionnels métier ayant une bonne connaissance des règles métier avec de nombreuses années d'expériences pouvaient être fortement impliqués dans le processus de développement d'applications d'entreprises. L'objectif étant ainsi d'offrir aux entreprises la meilleure approche possible pour le développement des solutions métier impliquant les systèmes automatisés. Un SI doit être capable de s'adapter aux changements de l'environnement. Mieux encore, il faudrait que la logique métier d'un SI puisse être modifiée par l'expert métier, sans avoir à attendre l'intervention des experts système. Ceci passe par la séparation de la logique métier (les RM) et la logique système (cf. figure 2.6) qui constitue le principe de base de l'ARM.

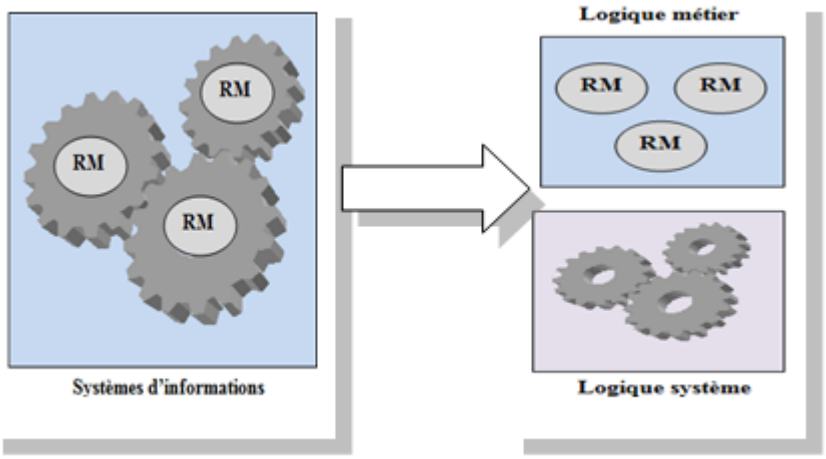


Figure 2.6: Séparation de la logique métier et système (Diouf, 2007)

Les règles métier permettent de séparer la logique métier (le comportement) et la logique système (le comment) d'une application. Ainsi, les experts système ne s'occupent que de la logique système tandis que les experts métier se chargent de la maintenance métier (qui change le plus souvent) dans un environnement «zéro-programmation» en utilisant des éditeurs en langage naturel (Diouf, 2007). Cependant, l'approche traditionnelle brièvement présentée plus haut ne permet pas une représentation appropriée des règles métier et l'adaptation aux changements reste trop coûteuse et peut mener à des erreurs graves.

Ayant pris conscience que les règles sont au cœur de toutes ces préoccupations qui est un actif (composant) essentiel pour l'entreprise, l'ARM a pour objectif d'externaliser la gestion des règles métier des applications métier. Elle cherche des moyens de recenser, de représenter et de gérer explicitement les règles métier de façon centralisée et rendues comme des services

externes au sein des systèmes d'information (Boyer et Mili, 2011). Cette gestion nécessite une application centralisée appelée système de gestion de règles métier (BRMS) ou moteur de règles qui est en charge de la gestion et du stockage de toutes les règles métier. En somme qu'est-ce qu'une approche par règles métier ?

"A formal way of managing and automating an organization's business rules so that the business behaves and evolves as its leaders intended" (Von Halle, 2001).

Cette définition est la plus utilisée car elle capture l'essence de l'ARM en une phrase comme suit :

- « *It is a formal approach* » : ceci veut dire que les processus, les taches et les rôles sont clairement définis, c'est-à-dire, une méthodologie.

- «*Managing and automating business rules* » : La gestion et l'automatisation sont reliées, mais restent des tâches distinctes : La gestion inclue la collecte, l'enregistrement, la validation (pour la précision), l'évaluation (valeur pour l'entreprise), l'édition et l'évolution des règles métier. Cette tâche doit être effectuée ou peut être effectuée, indépendamment du fait que les règles métier soient automatisables ou non. Pour ce qui est de l'automatisation, cela signifie de rendre ces règles métier opérationnelles c'est-à-dire, les représenter (via un langage exécutable) ou les transformer (via un interpréteur) pour les rendre exploitables par les applications métier.

- *[The business] behaves and evolves as [. . .] intended* : La logique métier doit être gérée et maintenue par les experts métier par opposition à l'approche traditionnelle afin de faire face à un environnement métier évolutif.

Les principes fondamentaux de l'ARM sont :

- La séparation nette de la logique métier et la logique système. La logique métier étant constituée d'un ensemble de règles métier qui doivent être externalisées des infrastructures informatiques et applications métier sous-jacentes afin d'être gérées de façon séparée et centralisée par les experts métier. Ainsi, les experts techniques (développeurs) ne s'occuperont que de la logique système applicative qui pourra appeler des règles métier comme des services externes.

- Les règles métier doivent être exprimées en langage naturel et de façon déclarative par les experts métier et être exécutées par un moteur de règle (BRMS) afin d'être utilisées dans

les SI. Ainsi, dans chaque domaine, les applications sont gérées par les experts métier du domaine : pour une application financière c'est l'expert financier, pour une application bancaire, c'est le banquier, etc.

2.3.3 En quoi l'approche par règles métier est-elle différente ?

Contrairement à l'approche traditionnelle décrite plus haut, l'introduction des règles métier et du BRMS par l'approche par règles métier apporte plusieurs changements dans le processus de modélisation des applications métier, sur les applications métier elles-mêmes, et finalement sur le domaine métier qui utilise ces applications.
La figure 2.7 montre l'architecture d'une application orientée règle métier.

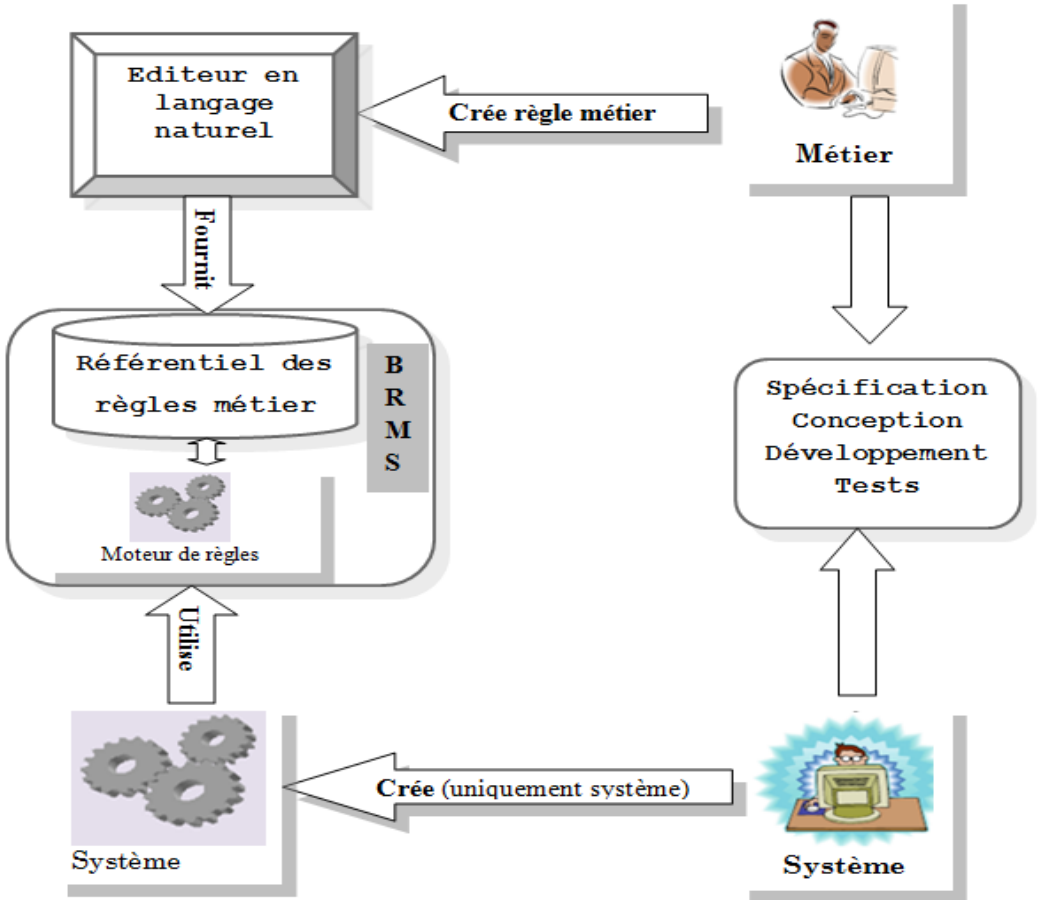


Figure 2.7 : Architecture d'un système orienté règle métier (Diouf, 2007)

Comme présenté sur la figure 2.7, l'expert métier travaille en étroite collaboration avec l'expert système pour la spécification et la conception de l'application. Durant cette phase, il peut être judicieux de mettre en place une ontologie commune afin d'éviter les ambigüités

dans le vocabulaire métier. Comme nous le verrons plus tard au Chapitre 5, cette ontologie est capitale pour la spécification des règles métier et représente le modèle objet métier qui reflète la vue du métier sur les données de l'entreprise. Les règles métier pertinentes pour les systèmes d'information sont gérées et exécutées à l'extérieur de ces applications métier par le BRMS. Un aspect important et clé de l'approche est que ces règles métier sont spécifiées de façon déclarative par les experts métier en utilisant un éditeur intuitif (en langage naturel). Elles sont ensuite stockées dans un référentiel de règles et gérées par le BRMS (le BRMS est décrit en détail à la Section 2.4.3). Le BRMS inclut un composant pour l'exécution des règles métier ou moteur de règles (*Rule Engine*) qui se charge d'exécuter des règles métier à la demande des applications métier. Les experts système se chargent de la logique applicative et des infrastructures informatiques qui interagissent avec le moteur de règles du BRMS. On constate ici qu'avec l'ARM, la logique métier ou décisionnelle est remplacée par l'invocation du moteur de règles du BRMS qui fournit le code du programme généré à partir de la règle métier explicitement définie par les experts métier. Ceci permet une séparation nette entre la logique métier et les applications métier qui constitue la clé de la traçabilité et de l'auditabilité de l'approche (Von Halle, 2001).

Considérons l'exemple de la règle métier suivante : «*Tout client Platinium qui achète plus de 500 articles aura 10% de remise*».
Ainsi avec un éditeur en langage naturel, un expert métier pourra entrer la règle comme présentée à la figure 2.8.

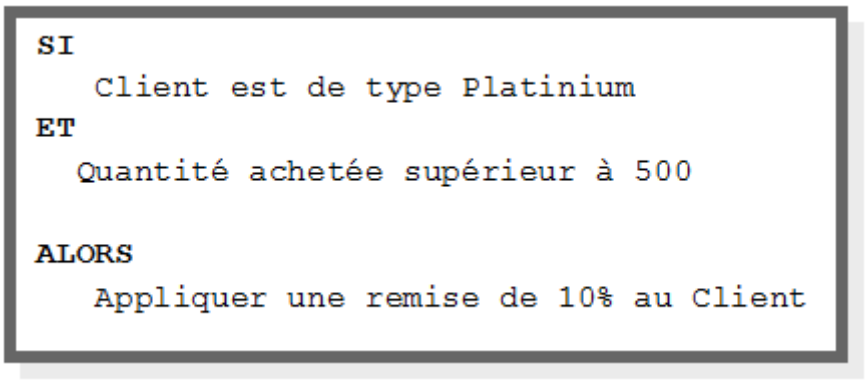


Figure 2.8: Exemple de règle dans un éditeur en langage naturel

En effet, les règles métier exprimées en langage naturel par les experts métier ne sont pas directement exécutables et de ce fait, ne peuvent pas être exploitées par les applications métier. Ainsi, elles doivent être d'abord transformées vers d'autres langages formels exécutables par les moteurs de règles. Ces langages formels sont la plupart de temps non compréhensibles par les experts métier. Cette transformation des règles métier du langage naturel vers les langages formels exécutable par le moteur de règles peut être prise en compte ou non par le BRMS. Une fois la règle transformée et représentée dans un langage formel, elle est stockée dans un référentiel de règles (*Business rule repository*) afin d'être exécutée plus tard par le moteur de règle du BRMS à la demande des applications métier.

La figure 2.9 montre l'exemple de notre règle métier citée plus haut transformée et représentée dans un formalisme XML.

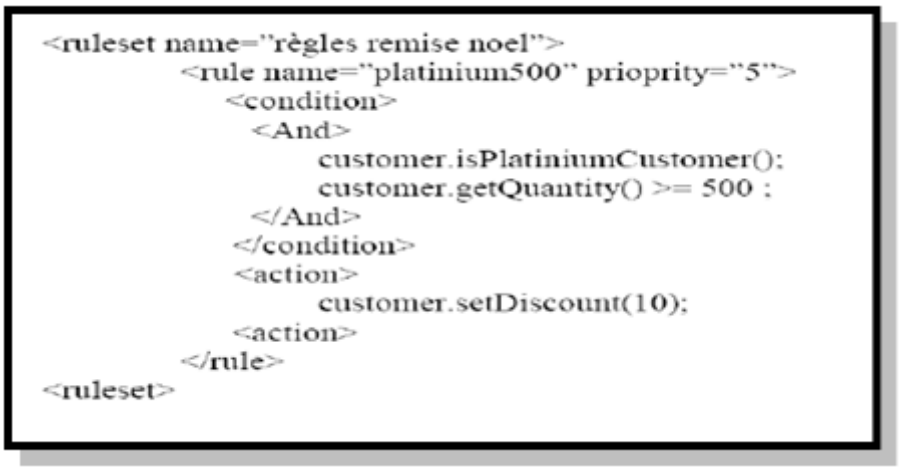


Figure 2.9: Exemple de règle dans un formalisme XML

Une bonne conception orientée objet doit généralement assigner à chaque fonction spécifique objet une interface qui à son tour va collaborer avec le moteur de règles pour produire le résultat. En considérant notre exemple cité plus haut, une bonne application métier orientée objet aura une méthode *setDiscount(10)* définie dans la classe Client qui retourne *true* si *isPlatiniumCustomer()* est *true* et *getQuantity()* >= 500 est aussi *true* et *false* dans le cas contraire. Dans une application traditionnelle, la méthode doit implémenter la logique métier décrite par la règle métier dans un langage de programmation (Java, C++ ou C#) de façon procédurale avec des boucles et des conditions if/then/else. Dans une application orientée règles métier, la logique métier (décisionnelle) doit plutôt être dans un langage de règle et son

exécution est déléguée au moteur des règles métier du BRMS. La bonne pratique est d'identifier les applications métier qui sont dépendantes des règles métier et interagissent avec le BRMS. En ce qui concerne le déploiement, une application orientée règle métier diffère d'une application traditionnelle dans le sens que l'approche applicative est divisée en deux parties : (1) Les règles métier qui sont gérées par un BRMS et (2) le reste des composants logiciels qui sont responsables des autres parties de l'application (la gestion des objets, des workflows, des services architecturaux, etc.). Ces deux parties sont packagées et déployées séparément.

La principale implication de l'ARM sur le métier est que les experts métier sont à la charge de la création et de la maintenance des règles métier. Ceci est l'une des motivations clés pour l'ARM sur l'approche traditionnelle qui a un besoin en agilité et en flexibilité face aux changements. Les applications orientées règles métier peuvent évoluer aussi vite que l'entreprise en a besoin. Le plus important est que les règles métier peuvent être modifiées à chaud[12] par les experts métier sans l'intervention des experts système comme le montre la figure 2.10. Ainsi, si la politique métier change et qu'un expert métier juge nécessaire de mettre à jour les règles métier, il lance son éditeur en langage naturel et fait les modifications qui sont ensuite stockées dans la base de règles. Les applications métier référencient toujours la base de connaissance mise à jour.

[12] Sans interrompre l'exécution des applications métier.

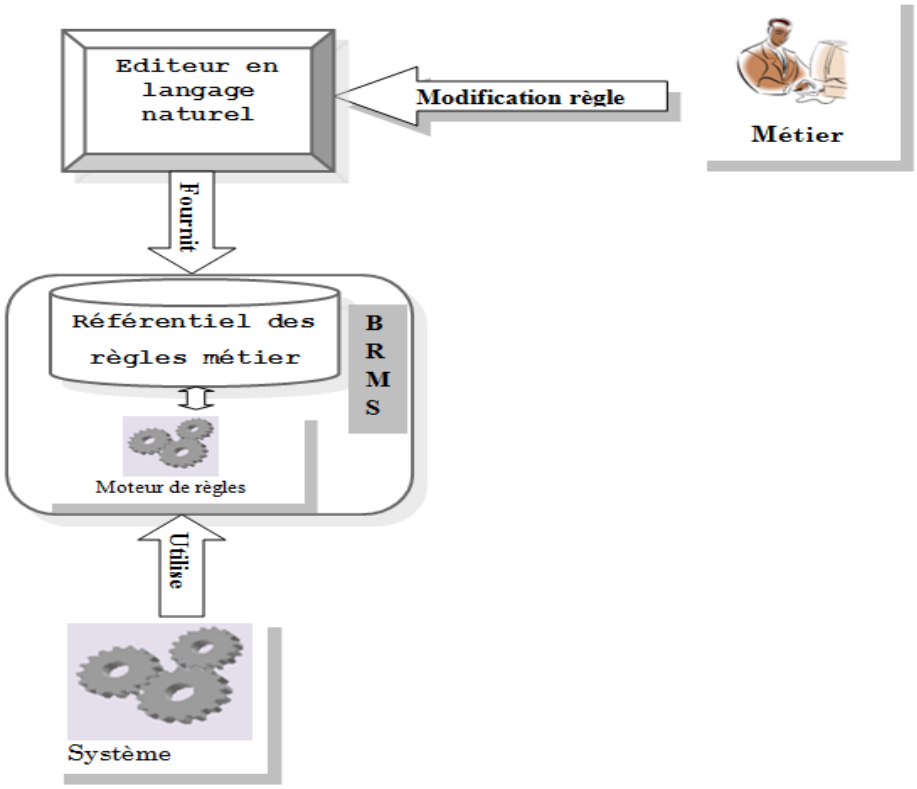


Figure 2.10 : Processus de modification des règles métier (Diouf, 2007)

Plusieurs facteurs rendent la maintenance plus facile et rapide (Boyer et Mili, 2011) :

- **La compréhensibilité par le métier :** les règles métier sont exprimées dans un langage que les utilisateurs métier peuvent comprendre, leur permettant soit de spécifier eux-mêmes leurs règles métier, soit de les valider facilement.
- **Le déploiement séparé :** puisque les règles métier sont déployées séparément du code source de l'application métier, nous pouvons avoir un cycle de maintenance et de déploiement des règles métier qui soit séparé du cycle de maintenance et du déploiement de l'application elle même. La figure 2.11 illustre les différents cycles de maintenance et de déploiement pour le code source des applications métier et pour les règles métier. La partie inférieure de la figure montre le cycle de maintenance et de déploiement pour le code de base de l'application métier, qui devrait être assez stable. Après un premier déploiement de l'application, on peut avoir une version de mise à jour ou deux dans la première année, mais après cela, le rythme de changement ralentit considérablement, souvent une fois ou moins. En ce qui concerne les règles métier, nous pouvons avoir de petites mises à jour aussi souvent que nécessaires, peut être tous les jours, etc.

- **L'exécution séparée :** De la même façon que le déploiement séparé, et comme aussi l'indique la figure 2.7 ci-dessus, les règles métier sont exécutées par le BRMS, à la demande des applications métier. Cela signifie que nous pouvons avoir un déploiement à chaud de nouvelles règles métier, sans arrêter l'application métier.

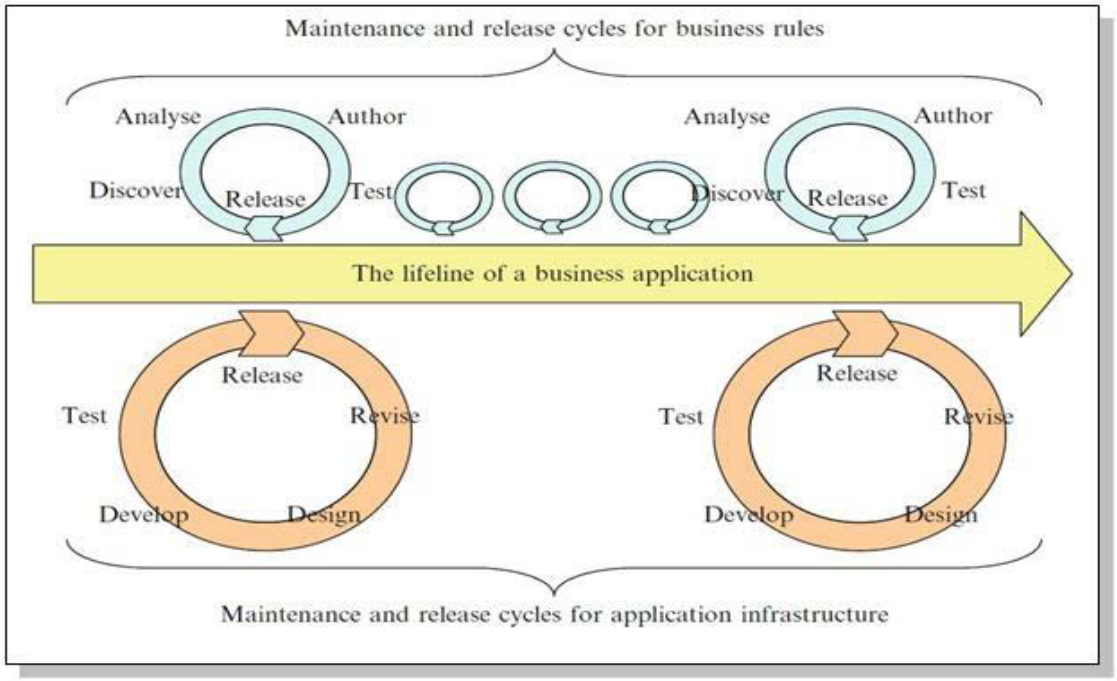


Figure 2.11: Maintenance and release cycles for application core versus business rules (Boyer et Mili, 2011)

2.4 Mise en œuvre de l'approche par règles métier

Au vu de ce qui précède sur les principes de l'ARM et de la figure 2.7 portant sur l'architecture d'un SI orienté règles métier, on peut déduire que le processus d'une mise en œuvre complète d'une approche par règle métier a trois (étapes) composants (Boyer et Mili, 2011) interconnectés (cf. figure 2.12) à savoir : une méthodologie, des langages et un outil appelé BRMS.

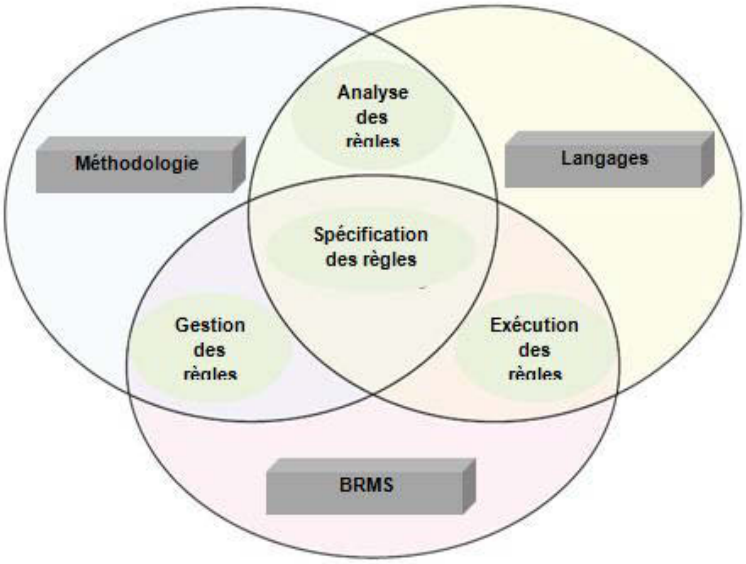


Figure 2.12 : Les trois composants d'une approche par règles métier et de leurs interrelations (Boyer et Mili, 2011)

2.4.1 Méthodologie

L'approche passe par une méthodologie claire pour la gestion des règles qui consiste en la collecte, l'analyse, la représentation, l'évaluation, la validation et l'évolution des règles métier.

2.4.2 Langages

L'approche définit un ou plusieurs langages plus ou moins formels pour l'expression des règles métier à différentes phases de leur cycle de vie et pour des différentes parties prenantes (experts métier, expert système et machines). Pour cela (Von Halle, 2001) propose quatre niveaux de langages à utiliser tout au long du cycle de vie des règles métier (cf. figure 2.13). Il convient ici de noter que l'un des principes clé de l'ARM est la spécification des règles métier par les experts métier. Ainsi, cette spécification dépend fortement du langage utilisé car il se doit d'être intuitif à ces derniers. Il doit être assez naturel (langage naturel structuré ou contrôlé) et formel (basé sur une grammaire formelle) afin d'être mappé vers des langages de règles exécutables. Sur les niveaux indiqués sur la figure 2.13, cela correspond à une fusion des deux catégories de langages du milieu (*Structured natural language* et *Formal authoring language*).

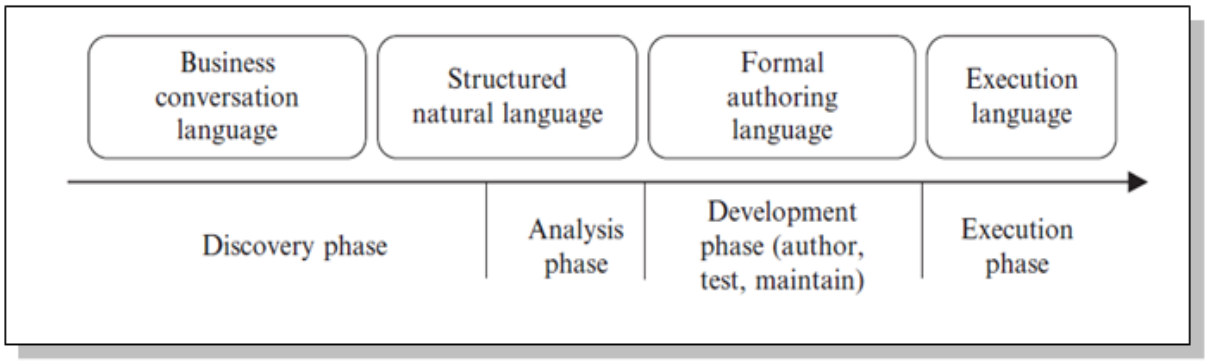


Figure 2.13 : Langages de règles en fonction de la phase du cycle de vie (Von Halle, 2001)

Comme nous avons déjà souligné à l'introduction, le langage métier (*Business conversation langage*) est similaire au langage naturel et donc par conséquent équivoque et ambigu. Les langages de règles exécutables (*Execution language*) sont par contre des langages de programmation difficile à utiliser par les experts métier. C'est donc au niveau de la fusion des deux langages du milieu que se situe notre langage contrôlé RuleCNL que nous proposons dans cette thèse comme indiqué par la figure 2.14.

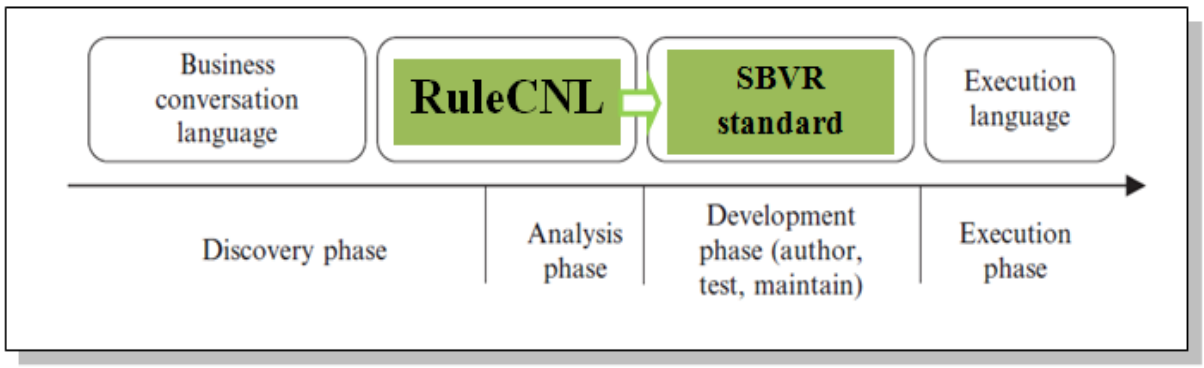


Figure 2.14 : Position de RuleCNL dans la mise en œuvre de l'ARM

2.4.3 Système de gestion des règles métier (BRMS)

Dans la pratique, l'ARM peut être mise en œuvre de plusieurs façons, mais la façon la plus courante est l'utilisation d'un système de gestion des règles métier appelé BRMS (Graham, 2006).

Il s'agit d'une application centrale avec un ensemble d'outils pour la gestion de processus/workflow, la gestion des rôles à travers les contrôles d'accès, etc.

L'objectif d'un BRMS est de faciliter la mise en place d'un système orienté règles métier en fournissant les outils nécessaires, allant de leur définition à leur exécution en passant par leur

stockage. Une caractéristique importante d'un BRMS est le stockage et la maintenance des règles métier (i.e. la logique métier) dans une base de règles de façon centralisée et partagée. Il permet la création et la modification des règles métier exprimées en langages de règles, et leur traduction vers d'autres langages exécutables.

Par analogie, nous pouvons comparer un BRMS à un système de gestion de base de données (SGBD) qui a pour objectif de rendre simple la gestion d'une base de données (Diouf, 2007). Dans l'ensemble, un BRMS complet doit pouvoir comporter trois composants essentiels comme indiqué sur la figure 2.15 : un composant de gestion (*Rule management*), un composant d'exécution (*Rule Automation*) ou moteur de règle et un référentiel de règles (*Rule repository*) partagées par les deux autres composants.

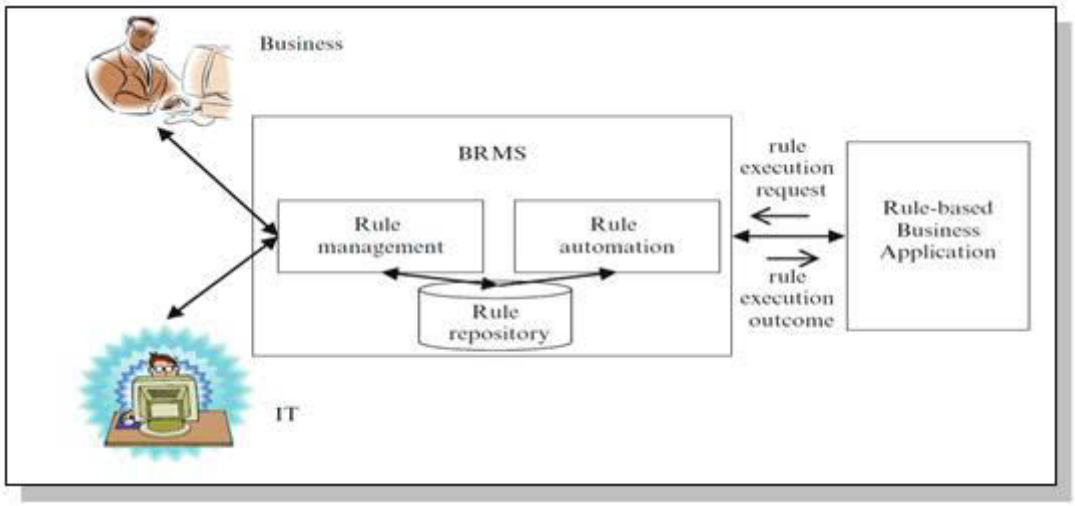


Figure 2.15: BRMS (Boyer et Mili, 2011)

Ainsi, nous pouvons déduire que le BRMS reste le cœur de l'ARM car une implémentation effective de l'approche dépend fortement des fonctionnalités offertes par le BRMS. Ces fonctionnalités sont décrites en détails ci-dessous.

2.4.3.1 Rule Management

C'est le composant qui offre des fonctionnalités générales pour la gestion des règles métier. Il doit pouvoir offrir des éditeurs et des outils pour permettre respectivement aux experts métier de spécifier et de maintenir leurs règles métier. La spécification des règles métier doit respecter plusieurs préalables :

(1) Cette spécification doit pouvoir se faire en utilisant un langage de règle intuitif et compréhensible par les utilisateurs métier (de préférence le langage naturel). Ainsi, ils pourront participer au développement, aux tests et aux processus de modification, réduisant ainsi la durée de l'implémentation et les erreurs d'interprétation entre l'intention (la volonté) du métier et la réalisation de celle-ci.
(2) Le langage de règle doit pouvoir permettre de spécifier la syntaxe et la sémantique de la règle.
(3) La spécification des règles métier doit être complètement indépendante des mécanismes utilisés pour manipuler les données et effectuer les actions. Le but est d'éviter un couplage fort entre les règles métier et l'application métier. Actuellement l'une des limitations de l'ARM comme nous avons souligné à l'introduction est qu'il n'existe pas de langage de règle standard. Dans la pratique, il existe une panoplie de BRMS avec des langages de règles techniques et propriétaires. Alors, si une entreprise utilisant un langage de règle « A » souhaite changer pour utiliser d'autres types de BRMS avec un autre langage « B », ces règles doivent être réécrites dans ce nouveau langage (Diouf et al., 2006). On comprend que si on est en face d'un grand nombre de règles, cela pose énormément de problème.
Seul un système d'écriture de règles métier standardisé offre la flexibilité et la transparence nécessaire pour fonctionner indépendamment de l'implémentation. C'est l'objectif poursuivi dans cette thèse car la sémantique de notre langage contrôlé RuleCNL est la transformation automatique vers le standard SBVR (cf. figure 2.14). À partir de là, on peut grâce aux transformations de modèles (cf. Chapitre 3) aller vers d'autres langages formels de règles spécifiques et aussi échanger des règles entre différents outils.

2.4.3.2 Rule automation ou Rule engine

L'un des principaux avantages d'avoir un BRMS complet est la réduction de la complexité de la mise en œuvre d'une logique applicative complexe pour permettre l'exécution de la logique métier. Le moteur de règles est le composant en charge de l'exécution des règles métier. La règle métier doit être représentée ici dans un langage exécutable avec une syntaxe et une sémantique bien définie. Le moteur de règles doit pouvoir fournir des interfaces bien définies aux applications métier via des API (*Application Programming Interface*) ou comme des services (*web services, service-oriented architecture (SOA)*) afin que les règles métier puissent être utilisées pour la prise des décisions au niveau du métier, en utilisant les données ordinaires du domaine métier. En conclusion, un moteur de règles prend comme entrée deux

paramètres comme indiquée par la figure 2.16 : la règle métier et la donnée métier. Le moteur de règles applique la règle métier sur les données en utilisant l'algorithme RETE (Stefik, 1995 ; Gupta et al. 1989 ; Forgy, 1982 ; Bunke et al. 1990 ; Jang et Tran 1990). Nous pouvons constater qu'un aspect clé et important de l'utilisation d'un moteur de règle est que les règles métier sont traitées aussi comme des données. Ceci implique deux choses : (a) elles peuvent être déployées séparément de l'application métier, (b) elles peuvent également être lues (appelées) aussi pendant l'exécution. Ceci prouve la flexibilité et l'adaptabilité de l'approche.

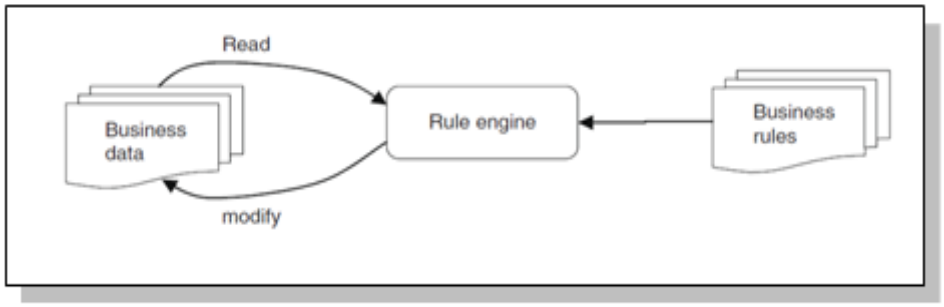


Figure 2.16 : Architecture d'un moteur de règle

Les moteurs de règles peuvent être subdivisés en deux catégories : ceux qui font un chaînage avant (*forward chaining*) et ceux qui font le chaînage arrière (*backward chaining*). Il en existe qui font les deux chaînages en même temps, dans ce cas on les appelle des moteurs hybrides (Diouf). Comprendre le mode de fonctionnement de ces chaînages est important pour comprendre pourquoi un moteur de règle est différent d'un autre et comment tirer le meilleur d'eux.

2.4.3.3 Rule repository

Elle sert de base de données pour les règles métier. Elle est partagée par les deux autres composants. La base de règles est en lecture et en modification par le composant de gestion, mais en lecture seule par le composant d'exécution (Boyer et Mili, 2011). Elle peut contenir plusieurs représentations de la même règle métier selon la phase du cycle de vie et l'audience concernée.

2.5 Conclusion

Les entreprises développent les systèmes d'information pour soutenir leur processus métier. Ces systèmes d'information doivent s'aligner avec la politique et les objectifs de l'entreprise. Ils le font par l'application des règles métier. En d'autres termes, les règles métier incarnent la logique métier et est au cœur des applications métier de l'entreprise. Les experts métier et les experts systèmes ont besoin de savoir quelles sont ces règles, et parfois les clients et les organismes de réglementation aussi. Les règles métier doivent être exprimées dans un langage que toutes les parties prenantes (métier, informatique, etc.) puissent comprendre, et doivent être implémentées de façon à permettre aux experts métier de les changer à la vitesse de l'évolution de leur métier, par opposition à la vitesse des infrastructures informatiques. Ce sont là les motivations derrière l'approche dite l'approche par règles métier que nous avons présenté dans ce chapitre. L'approche par règles métier dans la modélisation des SI promet d'apporter des gains significatifs en termes d'agilité face aux changements, de cohérence architecturale et de réutilisabilité de la logique métier. Ceci est basé sur un principe de base : la séparation des règles métier de l'infrastructure informatique sous-jacente.

L'approche de règles métier se compose de trois éléments interdépendants :

- Une méthodologie pour créer et gérer les règles métier.
- Un ou plusieurs langages pour les exprimer à différentes étapes de leur cycle de vie et pour différentes audiences.
- Un ensemble d'outils pour leur gestion et leur exécution à la demande des applications métier.

Les ingrédients de l'ARM existent depuis environs deux décennies. Il existe de plus en plus de moteurs de règles facilitant la mise en œuvre de l'ARM dans les entreprises. Ces moteurs de règle proposent des langages de règles formels pour l'exécution des règles métier pour des applications métier. Toutefois, ces langages formels sont difficilement manipulables par les experts métier eux-mêmes. Cependant l'un des principes clés de l'ARM est la spécification des règles métier par les experts métier eux même dans un langage intuitif. Nous avons vu dans la Section 2.4.2 qu'il existe au moins trois langages différents pendant le cycle de vie des règles métier : On a le langage naturel (métier) qui est d'abord utilisé au cours de conversations entre les utilisateurs métier pour décrire les règles, de façon informelle, sans

chercher à imposer une structure ou définir un modèle (*Template*) Ainsi, le langage métier ne peut être utilisé à cause de la redondance et l'inconstance dans les expressions de règles et dans les termes métier utilisés. À l'autre bout, on a des langages formels de règles qui sont des langages précis sans aucune ambigüité manipulables par des experts système. La règle se réfère exactement aux objets du système d'information. Ce langage est analysable et peut être exécuté par un moteur de règles. Il peut être de plusieurs types : la logique des prédicats, les tables de vérité ou les tables de décision qui présentent les règles comme ligne et colonnes représentant des conditions et des actions ou sous forme propriétaires à un moteur de règles. Pour réduire la distance sémantique entre ces deux langages (langage naturel du métier et langage formel de règles), il y a nécessité d'avoir un langage intermédiaire qu'on doit appeler langage contrôlé ou structuré ou dédié, etc. qui puisse être compris à la fois par des experts métier et les experts système. Le langage contrôlé RuleCNL que nous proposons dans cette thèse a donc pour but de réduire le pont entre le langage métier et les langages formels. Il a une syntaxe bien définie basée sur un méta-modèle et une grammaire et une sémantique qui consiste en des transformations vers les modèles du standard SBVR. Sa mise en œuvre suit une approche basée sur l'ingénierie dirigée par les modèles autour de ces deux principes clés : la méta-modélisation et la transformation des modèles. Dans le chapitre suivant, nous présentons tout d'abord les concepts fondamentaux de cette approche dirigée par les modèles autour de l'architecture MDA qui est un cadre normatif de l'OMG. Nous présentons ensuite le principe de la méta-modélisation et des transformations de modèles. Enfin, nous présentons le standard SBVR autour de son méta modèle qui est bien intégré dans l'architecture MDA.

Chapitre 3

Ingénierie dirigée par les modèles autour de la méta-modélisation et des transformations des modèles

3.1 Motivations pour l'ingénierie dirigée par les modèles

Face à la complexité sans cesse grandissante des systèmes d'information (distribués et de grande taille, etc.), la recherche autour du Génie logiciel qui est la discipline informatique visant à fournir des méthodes, techniques et outils concourant à la production d'un logiciel n'a cessé de croître. L'objectif récurrent est entre autre d'améliorer la qualité et la fiabilité des logiciels produits tout en augmentant la productivité et en minimisant le coût et les délais du processus de développement. Pendant des décennies, l'industrie du logiciel s'est trouvée confrontée à un certain nombre de challenges car de nombreux investissements dans le développement logiciel ont donné des résultats décevants. Beaucoup de projets ambitieux aboutissent à l'échec[13], d'autres vont jusqu'à dépasser largement le budget initial, certains systèmes qui réussissent d'abord se révèlent instables ou inflexibles au fil du temps (Hoffman, 1999)**.** Ainsi, le développement logiciel doit tenir compte non seulement de sa complexité, mais doit aussi relever de nouveaux défis liés à sa maintenance et son évolution. L'évolution est due aux changements importants et rapides qui interviennent à la fois au niveau de la logique métier (aspects fonctionnels) et de la logique système liée aux plateformes d'exécutions. Ainsi, afin de faire face à la complexité et aussi de maitriser ces changements, il est nécessaire que tout processus de développement sépare très clairement ces deux logiques. L'avantage est que les changements métier seront indépendants des changements technologiques et vice versa permettant ainsi à chacun d'évoluer à son propre rythme. Les changements métier permettront de répondre aux besoins métier facilement adaptables par les experts métier sans intrusions technologiques et les changements technologiques permettront de tirer profit de nouvelles plateformes d'exécutions. Dans ce contexte, on a assisté au cours

[13] "*85% of IT departments in the U.S. fail to meet their organizations' strategic business needs*." (Hoffman, 1999).

des décennies à une évolution des paradigmes de développement des SI. Le paradigme procédural, orienté objet, orienté composant sont des exemples de nouvelles approches qui ont aidé à cette quête. L'ingénierie dirigée par les modèles (IDM) ou MDE (*Model Driven Engineering*) tout en n'étant pas une panacée à tous ces problèmes, apparaît aujourd'hui comme le paradigme capable de fournir des réponses satisfaisantes à toutes ces nouvelles exigences (Hadj Kacem, 2008). L'IDM ne prétend pas remplacer les autres approches, mais cherche plutôt à les harmoniser dans une vision centrée sur les modèles afin d'améliorer la façon dont les SI sont développés (Hoffman, 1999). En effet, le concept de modèle n'est pas avant-gardiste en soi car il a été omniprésent dans le processus de développement, mais le concept innovateur est donc de faciliter la création d'opérations de production sur les modèles (i.e. de passer de l'ère des modèles purement contemplatifs à l'ère des modèles productifs) (Blanc, 2005).

L'adoption de l'IDM s'est fortement amplifiée avec le standard MDA de l'OMG décrivant le cadre normatif de l'utilisation des modèles au cours du processus de développement. Le MDA offre un mécanisme de modélisation à différents niveaux d'abstraction et à toutes les phases du cycle de vie des SI, permettant aux ingénieurs logiciels de se soustraire aux détails techniques des plateformes d'exécutions (J2ee, Net, PHP, etc.) pour se focaliser dans un premier sur des modèles (logique) métier indépendants de toutes plateformes d'exécutions et leur transformation par la suite vers plusieurs plateformes d'exécutions (Blanc, 2005). Ce chapitre a donc pour but de présenter à tour de rôle :

(1) Une vue macroscopique sur l'approche MDA autour de ces concepts de base et de son architecture.

(2) Les technologies clés autour du MDA que sont la méta-modélisation et les transformations de modèles qui sont les moyens/techniques que nous utilisons pour la mise en œuvre de notre langage contrôlé RuleCNL.

(3) Le standard SBVR autour de son méta-modèle sémantique nécessaire pour la définition de la sémantique de RuleCNL.

Toutefois, comme l'examen de toute discipline d'ingénierie doit commencer par son histoire, il est judicieux pour nous de faire tout d'abord un tour d'horizon sur l'évolution autour de quelques paradigmes de développement sur lesquels le MDA s'appuie.

3.2 Évolution des paradigmes de développement des systèmes d'information

Dès l'apparition des premiers ordinateurs, les instructions d'un programme devaient être écrites en langage machine (langage binaire) ou assembleur afin d'être exécutées par un ordinateur. Cette tâche était très fastidieuse car en plus de la description algorithmique d'un problème donné, le programmeur devrait avoir des connaissances minutieuses de la structure matérielle de l'ordinateur afin de pouvoir l'implémenter. Ceci a tout de suite fait naître un besoin de créer des langages intermédiaires, facilement compréhensibles et manipulables par les programmeurs. Ainsi, les programmes pourront être écrits dans ces langages et ensuite traduits ou compilés en langage machine pour exécution. De nos jours, dans le contexte du Génie logiciel, les techniques et les langages utilisés dans le processus de développement logiciel ont considérablement évolués. Cette évolution est dictée par le besoin de concevoir et de maintenir des logiciels toujours plus complexes. De nombreux langages ou paradigmes de développement se sont succédé dans le but de s'abstraire de plus en plus de la structure matérielle de bas niveau afin de faciliter le développement logiciel. On peut dire qu'ils nous ont affranchi des processeurs et même des systèmes d'exploitation. Chaque paradigme introduit des concepts, identifie une classe de problèmes et fournit des techniques permettant de les traiter. Nous résumons ci-dessous quelques uns des plus représentatifs.

3.2.1 Le paradigme procédural

Avant les années 80, on a assisté au paradigme procédural dont l'objectif était la séparation entre la représentation des données et leurs traitements. Ainsi, le concept de *structure de données* a été introduit pour représenter les données et les concepts de *procédures* et *fonctions* pour designer le traitement associé aux données. Ces concepts ont ensuite été implantés dans différents langages de programmation comme Pascal, Fortran, C, etc. Ce paradigme a très vite connu ses limites face aux applications de grande taille. En effet, l'interopérabilité entre les procédures/fonctions se fait par les variables globales qui sont accessibles par différentes procédures/fonctions. Ceci a engendré des effets de bord : si on oublie de mettre à jour certaines données partagées, si l'ordre chronologique des assignations par les différentes parties du logiciel est incorrect, ou si une zone de mémoire a été désallouée au mauvais moment, le programme se retrouve dans un état imprévu. Lors de la maintenance, la modification d'une procédure/fonction peut avoir des conséquences inattendues sur le résultat

d'autres procédures/fonctions liées par des variables globales. Ainsi, la maintenance des systèmes développés avec le paradigme procédural est difficile et coûteuse et la réutilisation du code se trouve souvent compromise. La nécessité d'avoir un paradigme un peu plus modulaire a donc donné place au paradigme orienté objet.

3.2.2 Le paradigme orienté objet

Vers les années 1980, le paradigme orienté objet a vu le jour pour pallier aux limites du paradigme procédural en introduisant le concept de *classe* et *d'objet*. Ce paradigme offre une meilleure protection des données car elles sont encapsulées dans un objet ainsi que leurs traitements. Ceci favorise également une meilleure réutilisation et extension grâce aux mécanismes d'héritage et de polymorphisme. Le paradigme orienté objet est implanté dans les langages comme C++, Java, etc. La modularité apportée par ce paradigme a nécessité également la conception de nouvelles méthodes de modélisation qui a conduit vers la définition et la standardisation d'un langage de modélisation objet unifié appelé UML. Cependant, malgré son essor, le paradigme orienté objet n'a pas réussi à surmonter toutes les difficultés liées au développement des applications distribuées et reparties compte tenu de sa faible granularité qui pose un problème de réutilisation des objets ou des classes. On va donc assister à une extension du paradigme orienté objet d'un point de vue architecture logicielle qui va conduire à un paradigme orienté composant.

3.2.3 Le paradigme orienté composant

Vers les années 1990 apparaît le paradigme orienté composant qui offre une granularité et un niveau d'abstraction élevé changeant ainsi la manière de développer les applications. Les applications sont désormais construites par assemblage de composants distribués et réutilisables, un composant étant une entité logicielle fait pour la composition (Barbier, 2005). L'objectif est d'industrialiser la réalisation des systèmes informatiques comme c'est le cas dans d'autres secteurs tel que l'automobile, le bâtiment, etc. Ainsi, l'approche orientée composant va présenter une vision plus macroscopique que l'approche objet par l'utilisation des intergiciels (*middleware*) comme EJB (*Enterprise JavaBeans*), CORBA (*Common Object Request Broker Architecture*), etc. Ces intergiciels définissent des abstractions pour les développeurs des applications réparties qui masquent les aspects d'implantation tels que le nommage, les transactions, la localisation, la sécurité et les communications sur réseaux hétérogènes. On assiste ainsi à une meilleure séparation des préoccupations. La conception

propose désormais de décomposer les applications en des aspects fonctionnels, traitant la logique métier de l'application et en des aspects non fonctionnels, traitant les détails techniques d'implémentation, délégués aux intergiciels. Cette séparation permet de réduire la complexité de conception, de réalisation et de maintenance des logiciels et d'en améliorer la compréhension, la réutilisation et l'évolution. Cependant, il existe une prolifération des intergiciels offrant divers services et sans cesse en évolution pour supporter de nouveaux services plus sophistiqués. Les entreprises sont ainsi tributaires des plateformes d'exécutions qui sont sans cesse en évolution. En plus, l'intégration de nouvelles fonctionnalités les oblige souvent de faire migrer leurs systèmes sur de nouvelles plateformes d'intergiciels et ceci à des coûts rédhibitoires. Une application étant dépendante de l'intergiciel sur lequel elle est bâtie, l'hétérogénéité inéluctable entre les différents intergiciels et le manque de standardisation rendent l'interopérabilité entre applications utilisant des intergiciels différents très complexe.

Afin de surmonter ce problème, il faut penser un nouveau paradigme basé sur une modélisation de haut niveau, centré sur la logique métier et indépendante des choix technologiques. Ce qui permet de garantir une pérennité des modèles conceptuels métier et la conception des applications portables tant au niveau des langages de programmation que des systèmes d'exploitation et aussi des intergiciels. Les entreprises peuvent ainsi gagner en productivité et en compétitivité, capitaliser les efforts consentis à tous les niveaux et tirer profit des avantages qu'offre la diversité des plates-formes. La continuité nous conduit vers l'IDM que nous avons mentionné plus haut dont l'intérêt a été fortement amplifié, lorsque l'OMG a rendu publique son initiative MDA qui vise à la définition d'un cadre normatif pour l'IDM.

3.3 Vue macroscopique sur le *Model Driven Architecture*

Chacun s'accorde à penser que l'approche MDA est l'avenir des applications à l'exemple des citations suivantes :

«*Modéliser est le futur, et je pense que les sociétés qui travaillent dans ce domaine ont raison*» Bill Gates, PDG de Microsoft.

«*Obtenir du code à partir d'un modèle stable est une capacité qui s'inscrit dans la durée*» Richard Soley, Directeur de l'OMG.

"*...MDA creates a huge upside for us and our customers*." --Richard Probst, Vice President, Enterprise Services Community, SAP.

Il y a également les pionniers du monde UML, comme Graddy Booch, Jim Rumbaugh ou Ivar Jacobson et bien d'autres. Tous affirment qu'il est temps d'élaborer les applications à partir de modèles et, surtout, de faire en sorte que ces modèles soient au centre du cycle de vie de ces applications, autrement dit qu'ils soient productifs.

Le MDA est une approche de modélisation proposée par l'OMG en 2001 en réponse au problème accru de proliférations des intergiciels et de l'interopérabilité entre les applications. Il n'est pas un changement radical dans la façon dont le processus de développement logiciel a évolué au fil des ans, mais plutôt une étape de l'évolution qui regroupe un certain nombre de tendances qui ont progressivement amélioré la façon dont nous produisons des logiciels (Frankel, 2003). Le MDA se base sur une idée bien connue et très vieille à savoir : la séparation des spécifications fonctionnelles d'un système, des spécifications de son implémentation sur une plateforme donnée**.** Ceci se traduit par l'élaboration des modèles de plus haut niveau d'abstraction et la définition des transformations paramétriques vers les plates-formes d'exécutions (Kleppe et al., 2003). La figure 3.1 présente le logo du MDA avec les différentes couches de spécifications (OMG, 2003a). Dans la première couche la plus interne, se trouvent les standards les plus importants de l'approche : UML, MOF et CWM (*Common Warehouse Metamodel*). Ces standards définissent l'infrastructure du MDA. Dans la couche suivante, on trouve quelques unes des plateformes supportées et également le standard XMI permettant les échanges entre les différentes plateformes. Dans la troisième couche, on trouve des services systèmes qui permettent de gérer les évènements, la sécurité, les répertoires et les transactions et enfin à l'extérieur, la dernière couche concerne les domaines pour lesquels des composants métier doivent être définis (*Domain Facilities*).

Le MDA est une approche qui augmente le pouvoir du modèle dans le développement des SI. Il est orienté modèle parce qu'il fournit un moyen d'utiliser les modèles afin de diriger le cours de la compréhension, de la construction, du déploiement, des opérations, de la maintenance et de la modification.

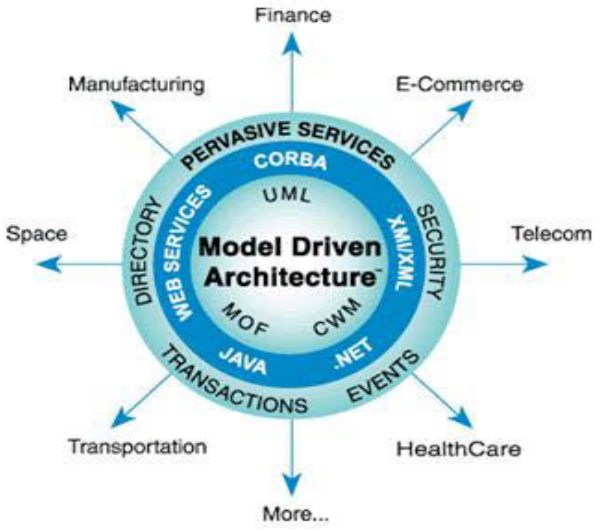


Figure 3.1 : *Le model Driven Architecture,* (OMG, 2003a)

Dans les approches basées sur les paradigmes décrits plus haut, les modèles sont produits dans le but de documenter l'application en cours de développement afin de faciliter la compréhension et la communication entre les différents acteurs de développement. Cependant, ces modèles ne sont pas automatisables (*computational models*) et n'ont pas de rôle productif. Ainsi, ils sont abandonnés pendant la phase de codage ou d'implémentation qui d'ailleurs constitue le cœur du processus de développement car le code est considéré comme le seul élément fiable du logiciel. Ainsi, des décisions conceptuelles peuvent être prises (et le code modifié) au cours de cette phase de codage sans que les modèles soient remis à jour (Blanc, 2005).

Dans l'approche MDA, il est désormais possible de capitaliser la définition d'une application en des modèles stables et de fournir un ensemble de transformations automatisées sur ces modèles vers différentes plateformes d'exécutions afin d'assurer la continuité entre la conception et le code. Ceci permet une séparation nette des spécifications des aspects fonctionnels d'un système des spécifications de l'implémentation de ces fonctionnalités sur une plateforme d'exécution spécifique (J2EE, .NET, etc.). Ainsi les applications ne sont plus dépendantes d'une technologie ou d'une plateforme spécifique et la logique métier peut être protégée contre les changements occasionnés par l'apparition d'une nouvelle technologie ou plateforme. La portabilité et l'interopérabilité des applications sont déportées au niveau des modèles. On passe ainsi de l'interopérabilité du code vers l'interopérabilité des modèles ou encore du «*middleware*» au «*modelware*». Les trois principaux objectifs du MDA sont donc

la portabilité, l'interopérabilité et la réutilisabilité grâce à une séparation nette des préoccupations (OMG, 2003a).

3.3.1 Les concepts de base

Cette section définit les concepts de base du MDA et les relations qui existent entre eux (Muller, 2006). L'approche MDA définit une architecture à quatre couches comme indiqué sur la figure 3.2 autour des modèles qui fournie différents niveaux d'abstraction pour représenter tout système et les différents modèles/langages qui le décrivent. Cette architecture est constituée au niveau le plus bas, par la couche M0 représentant le système réel (application informatique), le niveau M1 contient les différents modèles du système, le niveau M2 contient tous les méta-modèles qui ont été utilisés pour définir les modèles, et le niveau M3 contient le méta- méta-modèle qui a permis de définir uniformément les méta-modèles.

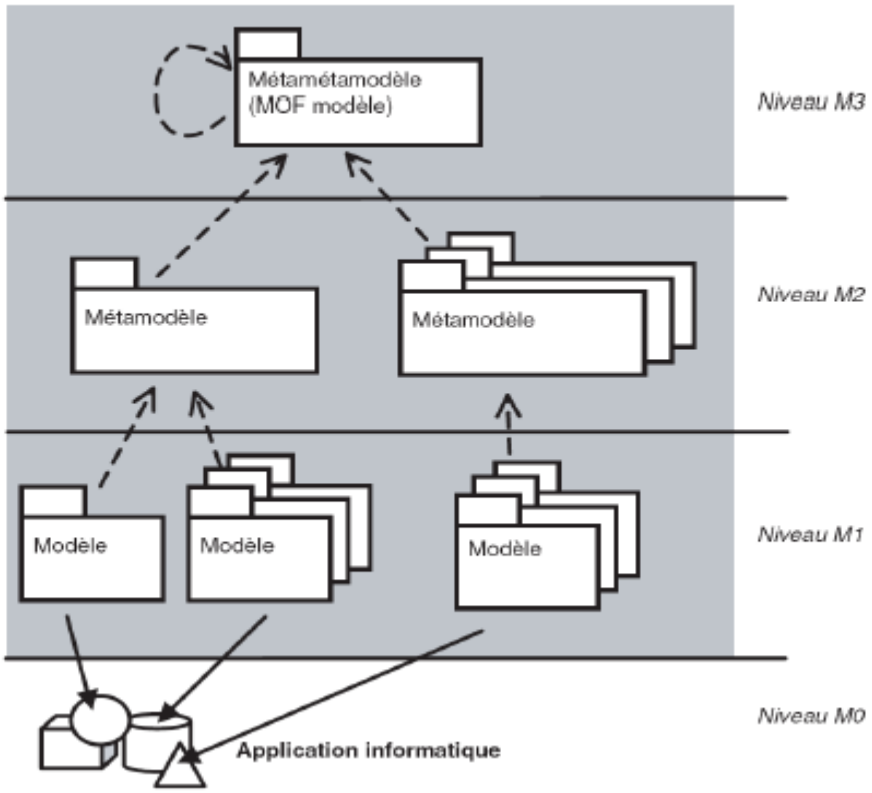


Figure 3.2 : Architecture à quatre niveaux du MDA

3.3.1.1 Modèles et systèmes

Le concept central du MDA est la notion de modèle. Quelque soit la discipline scientifique considérée, un modèle est une abstraction d'un système construit dans un but précis. On dit alors que le modèle représente le système (programmes, applications informatiques, etc.). Un modèle est une abstraction dans la mesure où il contient un ensemble restreint d'informations sur le système qu‟il représente. Il est construit dans un but précis et les informations qu'il contient sont choisies pour être pertinentes vis-à-vis de l'utilisation qui sera faite du modèle. Un modèle est souvent présenté sous la forme d'une combinaison de schémas et de textes. Le texte peut être écrit avec un langage de modélisation ou avec un langage naturel.

3.3.1.2 Méta-modèle

Afin de rendre un modèle utilisable (productif), il est nécessaire de préciser le langage dans lequel il est exprimé. En plus ce langage doit être clairement défini afin que les modèles soient manipulés par les machines. On utilise pour cela un méta-modèle. Un méta-modèle est un modèle qui définit le langage d‟expression d‟un modèle c'est-à-dire le langage de modélisation (OMG, 2006a). Le concept de méta-modèle permet de définir les caractéristiques communes à un ensemble de modèles. Un méta-modèle représente une spécification formelle d‟une abstraction (syntaxe abstraite), généralement consensuelle et normative d‟un langage de modélisation.

Un modèle est lié à son méta-modèle par une relation de *conformité*. On dit qu'un modèle est conforme à un méta-modèle si l'ensemble des éléments du modèle est défini par le méta-modèle. Cette notion de conformité est essentielle à l'ingénierie des modèle mais n'est pas nouvelle : un texte est conforme à une grammaire, un programme JAVA est conforme au langage Java et un document XML est conforme à sa DTD/XML Schéma.

3.3.1.3 Méta-métamodèle

De la même manière qu'il est nécessaire d'avoir un méta-modèle pour interpréter un modèle, pour pouvoir interpréter un méta-modèle il faut disposer d'une description du langage dans lequel il est écrit : un modèle pour les méta-modèles. C'est naturellement que l'on désigne ce modèle particulier par le terme de méta-métamodèle. De par sa position dans la hiérarchie

d'utilisation, le choix du méta-métamodèle est très important, car de lui vont dépendre tous les méta-modèles et modèles définis par la suite.

Afin de se soustraire au problème de la définition des méta-méta-modèles (et ainsi d'éviter d'avoir à définir un méta-méta-méta-modèle), l'idée généralement retenue est de concevoir les méta-méta-modèles de sorte qu'ils soient auto-descriptifs, c'est-à-dire qui se définissent eux-mêmes. L'approche MDA définit le MOF comme l'unique méta-métamodèle pour la définition des différents méta-modèles.

3.3.1.4 Notion de plateforme

Dans l'approche MDA, la notion de plateforme revient souvent. Selon le guide MDA proposé par l'OMG, une plateforme est définie comme un ensemble de sous-systèmes et de technologies, qui fournit aux applications supportées, un ensemble cohérent de fonctionnalités à travers des interfaces et des gabarits masquant leurs détails d'implémentation (OMG, 2003a). Il faut noter que cette définition reste générale et que la notion de plateforme varie selon les domaines.

3.3.2 La mise en œuvre du *Model Driven Architecture*

Le principe clé du MDA est l'utilisation de modèles aux différentes phases du processus de développement d'un SI. Son objectif majeur est l'élaboration de modèles pérennes, indépendants des détails techniques des plates-formes d'exécution (J2EE, .Net, etc.), afin de permettre la génération automatique de la totalité du code des applications et d'obtenir un gain significatif de productivité (Blanc, 2005). Ce découpage permet ainsi de séparer les spécifications fonctionnelles d'un système des spécifications de son implémentation sur une plateforme donnée et de déployer le même modèle sur plusieurs plateformes grâce à des projections standardisées (cf. figure 3.3) (OMG, 2003a). Il permet aux applications d'inter-opérer en reliant leurs modèles et supporte l'évolution des plateformes et des techniques (Accord, 2002). La mise en œuvre du MDA est entièrement basée sur les modèles et leurs transformations et consiste à définir des modèles d'exigences ou des modèles métier (*CIM*) dans lesquels aucune considération informatique n'apparait, des modèles d'analyse et de conception (*PIM*) indépendants de toutes plateformes, des modèles de code (*PSM*) dépendants de la plateforme pour l'implémentation concrète du système (OMG, 2003a).

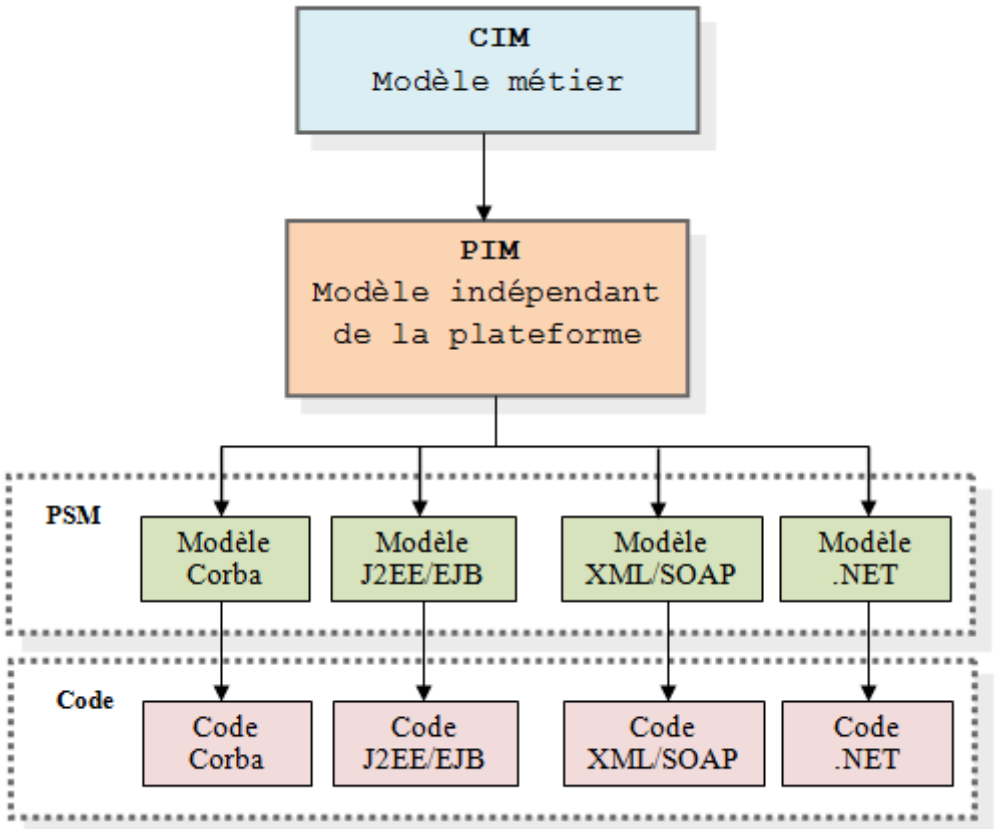


Figure 3.3 : Les modèles du MDA

3.3.2.1 *Computation Independent Model*

La première chose à faire lors de la construction d'une nouvelle application est bien entendu de spécifier les exigences du client. Bien que très en amont, cette étape doit fortement bénéficier des modèles. Nous supposons que l'utilisateur premier, l'expert métier, ne connaît rien en modèles et artefacts utilisés pour réaliser les fonctionnalités du CIM. Le CIM joue un rôle très important dans le rapprochement entre ceux qui sont experts du domaine et des besoins d'un coté et ceux qui sont experts en design et constructions de l'autre. Il est important de noter qu'un modèle d'exigences ne contient pas d'informations sur la réalisation de l'application ni sur les traitements.

Il faut noter que le MDA a spécifié les modèles CIM par souci d'exhaustivité, car il n'émet aucune préconisation quant à l'élaboration des modèles métier. Ces modèles restent néanmoins beaucoup moins utilisés dans la pratique par la communauté MDA que les PIM et les PSM. C'est ainsi à ce niveau que se situe notre RuleCNL qui est un langage contrôlé dédié

aux experts métier pour l'expression de leur vocabulaire et règles métier et des transformations automatiques vers des modèles sémantiques SBVR.

3.3.2.2 *Platform Independent Model*

Cette étape constitue en pratique la première étape du processus MDA et consiste à réaliser des modèles indépendants de toutes plateformes (PIM), exprimés en UML. Il existe plusieurs niveaux de PIM mais tous sont indépendants de n'importe quelle plateforme d'exécution.
Le PIM de base représente uniquement les capacités fonctionnelles métier et le comportement du système, sans dégradations par des considérations technologiques. La clarté de ce modèle doit permettre à des experts du domaine de le comprendre bien mieux qu'un modèle d'implémentation. Ils peuvent ainsi vérifier plus facilement que le PIM est complet et correct. Un autre bénéfice de l'indépendance technique de ce modèle est qu'il garde tout son intérêt au cours du temps et doit être modifié uniquement si les connaissances ou les besoins du métier changent.

D'autres PIM intègrent des aspects technologiques et architecturaux mais toujours sans détails spécifiques à une plateforme. Ces modèles peuvent, par exemple, contenir des informations sur la persistance, les transactions, la sécurité, etc. Ces concepts permettent de projeter plus précisément le modèle PIM vers un modèle spécifique (PSM).

3.3.2.3 *Platform Specific Model*

Une fois un modèle PIM suffisamment détaillé, il est projeté vers un modèle spécifique (PSM). Pour obtenir un modèle spécifique, il faut choisir la ou les plateformes d'exécutions (plusieurs plates-formes peuvent être utilisées pour mettre en œuvre un même modèle). Les caractéristiques d'exécution et les informations de configuration qui ont été définies de façon générique sont converties pour tenir compte des spécificités de la plateforme.

Comme pour les PIM, il existe plusieurs niveaux de PSM. Le premier, sous forme d'un schéma UML, est obtenu directement à partir du modèle PIM, les autres sont obtenus par transformations successives jusqu'à l'obtention du système exécutable.

3.4 La mise en œuvre de RuleCNL autour des principes clés du MDA

Dans l'approche MDA, l'introduction du concept central de *modèle* a suscité bien évidemment la définition d'un langage de modélisation encore appelé méta-modèle (Blanc, 2005). Ainsi, on doit donc pouvoir définir différents langages de modélisation pour spécifier les modèles aux différents niveaux d'abstraction dans l'architecture MDA à savoir les CIM, les PIM et les PSM. Chaque langage de modélisation doit définir des concepts ainsi que les relations entre eux appropriés pour le niveau d'abstraction considéré. Normalement, les modèles métier ou CIM doivent avoir leur propre langage de modélisation (méta-modèle) très proche du langage métier (naturel), les modèles PIM leur propre méta-modèle (formel) ainsi que les modèles PSM (Blanc et al., 2005). En effet, dans la pratique du MDA comme souligné dans la section précédente et au Chapitre 1, les modèles CIM ne sont pas pris en compte car la communauté MDA n'émet aucune préconisation quant à l'élaboration de ces modèles métier. Le langage de modélisation UML a été initialement le méta-modèle préconisé par l'OMG pour spécifier les PIM et la plupart des PSM. Ce choix s'est justifié par le fait qu'avec l'apparition du paradigme objet (cf. Section 3.2.2), UML était devenu le standard incontournable des langages de modélisation objet. Il était déjà adopté et implanté dans les entreprises et supporté par une armada d'outils. Les nouveaux apports de sa version 2.0 l'ont placé ainsi au centre du MDA et a fait de lui le méta-modèle le plus important de cette approche (OMG, 2003a). Cependant, au fil du temps, UML restant un mal nécessaire, ne sera plus la panacée de modélisation en raison de son approche généraliste. Il n'offre pas assez de concepts spécifiques pour prendre en charge certains domaines d'applications ou métier. Malgré les mécanismes d'extensibilité d'UML basés sur les profils (OMG, 2007 ; Czarnecki et Helsen, 2003), force est de constater qu'UML ne peut couvrir tous les contextes. Les mécanismes d'extensibilité n'ont pas tous la précision souhaitable et mènent parfois à des contorsions dangereuses pour rester dans le monde UML. Pour pallier à ce manque, l'OMG va chercher des solutions visant à promouvoir des langages de modélisation dédiés, contrôlés, spécialisés ou « *sur mesure* »[14], etc. à un domaine particulier encore appelés DSL (*Domain Specific Language*). Ces langages dédiés offriront ainsi aux utilisateurs des concepts propres à leur métier et dont ils ont la maitrise. Cette nouvelle vision de l'OMG a donc constitué un aspect clé de l'approche MDA connu sous le nom de méta-modélisation (*Metamodeling*).

[14] Dans la littérature, selon les domaines, on trouve plusieurs dénominations des DSL.

3.4.1 La méta-modélisation

La méta-modélisation est l'activité qui consiste à définir le méta-modèle d'un langage de modélisation (OMG, 2007 ; Combemale, 2008). Elle vise donc à bien modéliser un langage, qui joue alors le rôle de système à modéliser. Notons qu'après l'introduction du concept clé de méta-modélisation comme processus de description des méta-modèles eux mêmes, de nombreux méta-modèles ont émergés afin d'apporter chacun leurs spécificités dans un domaine particulier (développement logiciel, entrepôt de données, procédé de développement, etc.). Devant le danger de voir émerger indépendamment et de manière incompatible cette grande variété de méta-modèles, il y a eu un besoin urgent de donner un cadre général et normatif pour leur description. La réponse logique fut donc d'offrir un langage standard de définition de méta-modèles qui prit lui-même la forme d'un modèle : ce fût le méta-métamodèle MOF (OMG, 2006a) proposé par l'OMG dans l'approche MDA comme standard de méta-modélisation. En tant que modèle, il doit également être défini à partir d'un langage de modélisation. Pour limiter le nombre de niveaux d'abstraction et enrayer la montée dans les niveaux méta, il doit alors avoir la propriété de méta-circularité, c'est-à-dire la capacité de se décrire lui-même. Ainsi l'OMG a défini le MOF de telle sorte qu'il soit auto descriptif.

Dans l'approche MDA, le MOF est unique et constitue un pilier de valeur car c'est le méta-métamodèle permettant de spécifier tous les méta-modèles. Même les méta-modèles standards de l'OMG comme UML (OMG, 2007), CWM (OMG, 2003b), SPEM (OMG, 2005) sont conformes au MOF. Bien qu'il ne puisse pas prétendre être le langage de méta-modélisation historique, le succès de MOF comme langage de méta-modélisation est si large qu'il est difficile de trouver une approche vraiment différente de nos jours (Fondement, 2007). D'autres recherches et outils sont des sous-ensembles et/ou extensions de MOF comme EMF (Budinsky et al., 2003), Kermeta (Jouault et Bézivin, 2006 ; Atlas, 2005), etc.

Un méta-modèle MOF (méta-modèle conforme au standard MOF) est considéré comme la spécification de la syntaxe abstraite d'un langage, à savoir le lexique des concepts du langage (vocabulaire), les propriétés de ces concepts, et les relations entre ces concepts (Fondement, 2007). À coté de cette syntaxe abstraite s'ajoute souvent des spécifications textuelles informelles (contraintes) portant sur la sémantique du langage. La syntaxe abstraite est généralement décrite par un diagramme de classe avec des notations UML avec des contraintes exprimées en langage OCL (*Object Constraint Language*) (OMG, 2006b).

Le grand avantage apporté par MOF est que tous les méta-modèles sont structurés de la même manière. Cette structuration commune des méta-modèles MOF permet de définir des mécanismes génériques à l'instar des transformations de modèles qui s'opère au niveau des méta-modèles et constitue aussi l'un des principes clés du MDA. Le standard MOF définit aussi des projections entre les méta-modèles MOF et d'autres plateformes techniques comme XML à travers le standard XMI (OMG, 2005b) utilisé pour l'échange de modèles et Java à travers le standard JMI (*Java Metadata Interface*) (Java Community Process, 2002) qui définit une API Java pour la manipulation de modèles suivant l'approche orienté objet. Dans le cas du standard XMI, les méta-modèles sont représentés dans un XML Schéma (DTT ou XSD) tandis que les modèles sont représentés dans un fichier XML.
La figure 3.4 illustre une partie de ce que pourrait être un diagramme de classes représentant MOF 1.4.

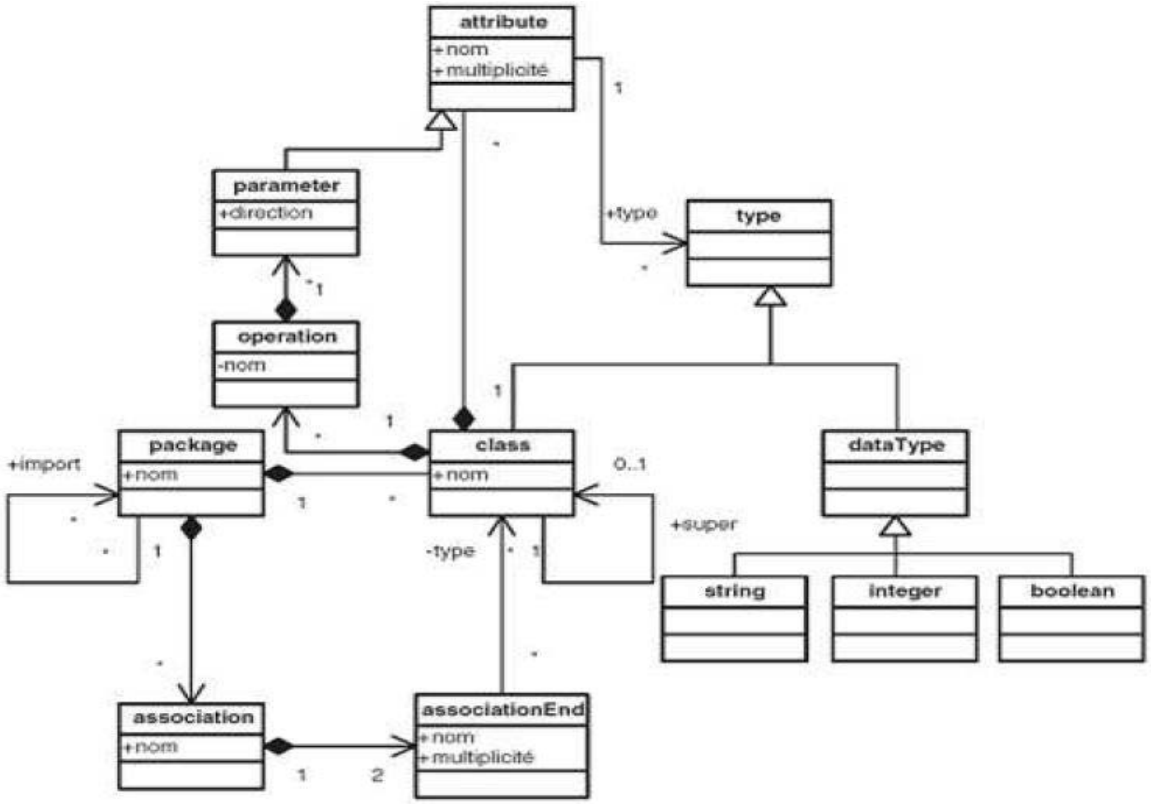


Figure 3.4 : Représentation de MOF 1.4 sous forme de diagramme de classe

MOF est ainsi considéré comme le standard pour la définition des méta-modèles, c'est-à dire pour modéliser la syntaxe abstraite des langages contrôlés ou langages dédiés. MOF joue exactement le rôle que joue EBNF pour définir les grammaires des langages formels. C'est

ainsi sur ce principe de méta-modélisation définit par MOF que nous avons défini le méta-modèle de notre langage contrôlé RuleCNL comme le montre la figure 3.5.

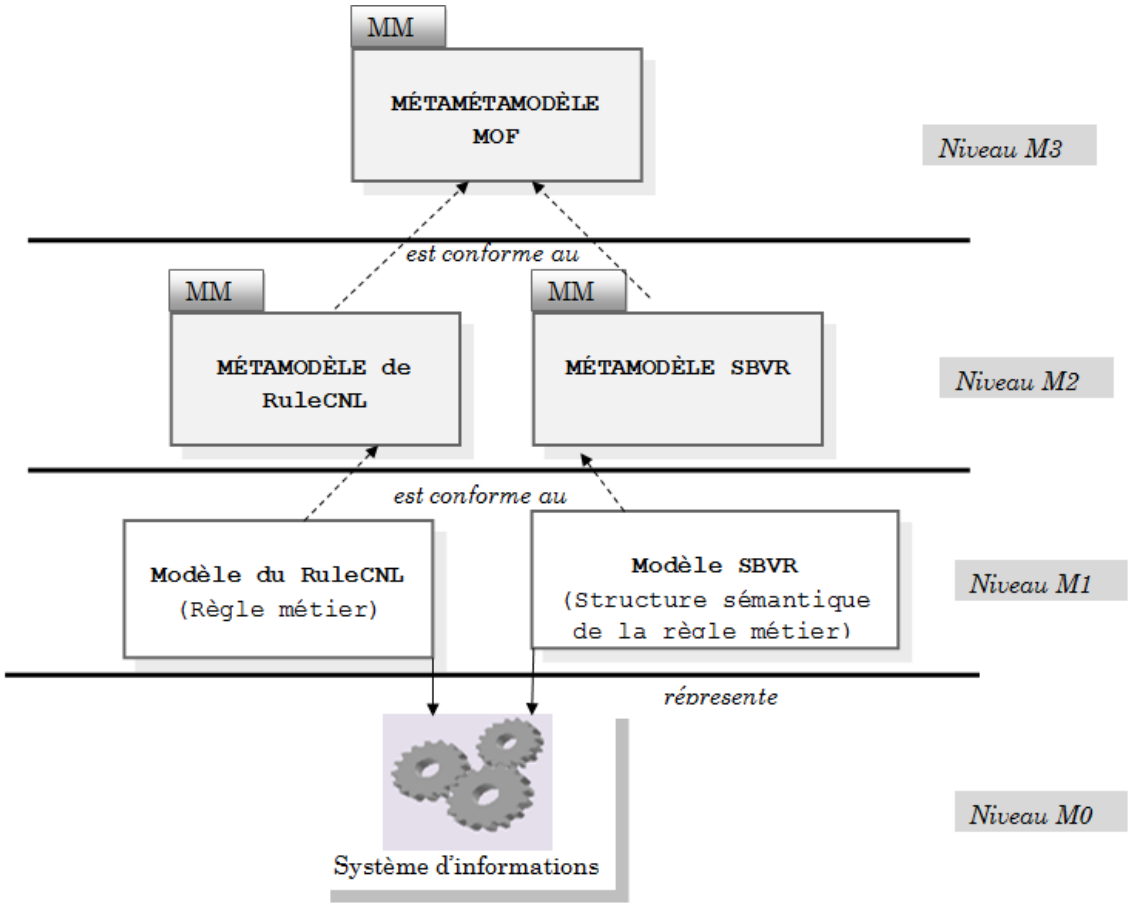


Figure 3.5 : Notre RuleCNL dans l'architecture à quatre niveaux de l'OMG

3.4.2 Les transformation des modèles

Nous avons passé en revue dans la Section 3.2.2 les trois types de modèles les plus importants pour MDA que sont les CIM, PIM et PSM. Ainsi il est important de bien établir les liens de traçabilité entre ces différents modèles. En fait, le MDA établit ces liens grâce au principe des transformations des modèles qui est un autre aspect clé de l'approche.

L'ère du MDA permet le passage d'une approche interprétative (où les modèles produits sont contemplatifs) à une approche transformationnelle (où les modèles produits sont productifs) (Blanc, 2005). Dans l'approche interprétative, l'individu a un rôle actif dans la construction

des systèmes informatiques alors que dans l'approche transformationnelle, il a un rôle simplifié et amoindri grâce aux constructions et transformations automatisées. D'un point de vue général, on appelle transformation de modèles tout programme dont les entrées et les sorties sont des modèles. Il faut vraiment insister sur leur importance, car elles sont au cœur des aspects de production du MDA, elles représentent en général la base de la production des systèmes dirigés par les modèles et sont le support de l'interopérabilité entre ces systèmes (Nebut et Falleri, 2006).

Le succès de l'IDM en général et du MDA en particulier repose en grande partie sur la résolution du problème de transformation. Ainsi, l'OMG veut proposer à terme l'automatisation de ces transformations par des outils logiciels. Il sera alors possible de passer du modèle d'analyse au déploiement de la solution en générant automatiquement du code. Mais dans un premier temps, ce passage de modèle à modèle va s'effectuer, de façon semi-automatique ou assisté, puis progressivement de manière partielle pour arriver enfin à une génération totale du code. Ces transformations seront de plus en plus automatisées au fur et à mesure de l'évolution des outils.

Les transformations de modèles préconisées par MDA sont essentiellement les transformations PIM vers PIM ou PSM vers PSM pour garder le même niveau d'abstraction (transformation horizontale) et PIM vers PSM pour changer le niveau d'abstraction (transformation verticale). La génération de code à partir des PSM est aussi primordiale et le MDA envisage aussi les transformations inverses : code vers PSM et PSM vers PIM. Quand le méta-modèle source est différent du méta-modèle cible, on parle de transformation exogène et dans le cas contraire, on parle de transformation endogène (Combemale, 2008). Ces différentes classes de transformations sont représentables sur la figure 3.6.

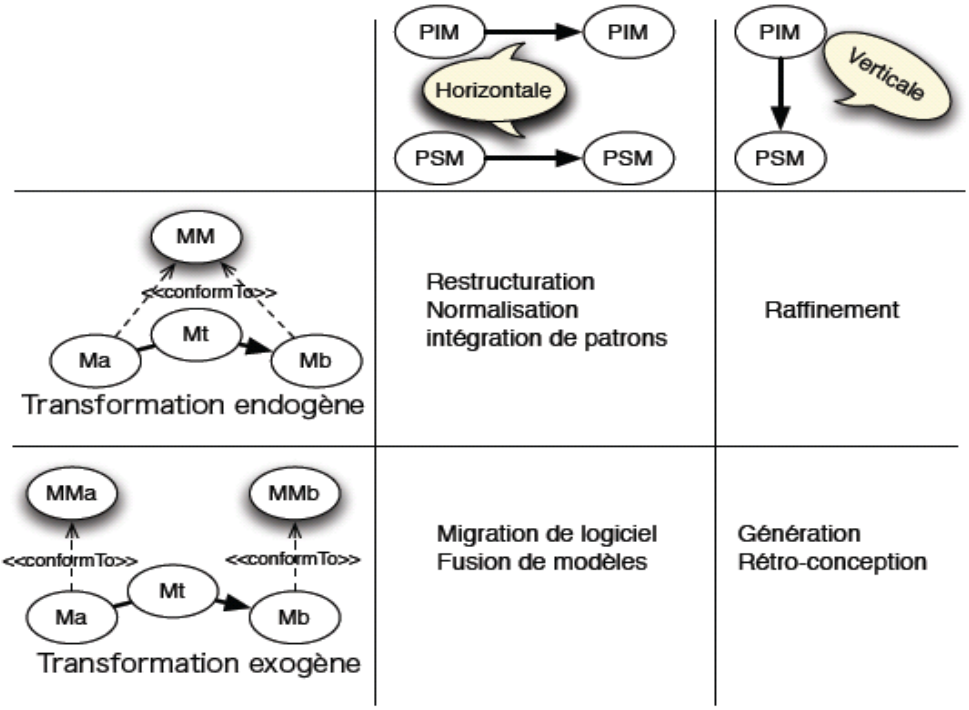


Figure 3.6 : Types de transformations et leurs principales utilisations (Combemale, 2008)

Les transformations de modèles peuvent s'écrire selon trois approches (Blanc, 2005) : Programmation, Template et Modélisation. Dans l'approche par programmation, la transformation est un programme qui utilise un API de manipulation des modèles à l'instar de l'API Java de l'EMF. Dans l'approche par template, la transformation est un template écrit dans un langage dédié à l'instar du template Acceleo[15]. L'approche par Modélisation est d'ailleurs celle préconisée par le MDA et c'est celle que nous utilisons pour la définition de la sémantique de RuleCNL qui est basée sur une approche transformationnelle. Elle est présentée plus en détails dans la section suivante.

3.4.2.1 Transformation basée sur la modélisation

Le mécanisme de transformation de modèles basé sur la modélisation est un processus à trois phases à savoir la définition des règles de transformation, l'expression des règles de transformation et l'exécution des règles de transformation.

[15] https://www.eclipse.org/acceleo

- **Définition des règles de transformation**

Étant donné un modèle source dans un langage Ls (tel que UML), et un modèle cible dans un langage Lc (tel que Java), il s‟agit dans cette étape d‟élaborer une mise en correspondance des concepts de Ls à ceux de Lc (ex. une classe UML correspond à une ou plusieurs classes Java). Dès lors, on fait recours aux techniques de la méta-modélisation présentée plus haut pour mettre en place une base de règles exhaustive et générique**.**

Les règles de transformation sont établies entre le méta-modèle source et le méta-modèle cible, c'est-à-dire entre l‟ensemble des concepts du modèle source et celui du modèle cible. Le processus de transformation prend en entrée un modèle conforme au méta-modèle source et produit en sortie un ou plusieurs autre(s) modèle(s) conforme(s) au méta-modèle cible, en utilisant les règles préalablement établies (cf. figure 3.7).

- **Expression des règles de transformation**

Pour définir une transformation de modèles, on peut utiliser un langage non formel, un langage d‟action pour représenter l‟algorithme de la transformation, ou encore un langage bien défini de mapping de modèles. Conscient du besoin d‟un langage bien défini et normalisé pour l‟expression des règles de transformations de modèles, l‟OMG a émis un appel à proposition (*RFP : Request For Proposal*) pour MOF2.0 QVT (OMG, 2008b) pour la standardisation du processus de transformation. Ce standard exige que les transformations de modèles soient définies avec précision en termes de rapport entre un méta-modèle source et un méta-modèle cible et que ces deux méta-modèles soient tous conformes à MOF.

- **Exécution des règles de transformation**

Une fois spécifiées et exprimées, les règles requièrent un moteur d‟exécution pour être exécutées. Ce moteur prend comme entrée le modèle et le méta-modèle source, le méta-modèle cible, ainsi que le modèle de transformation (les règles de transformation écrites dans le langage de transformation, basées sur les correspondances entre les deux méta-modèles source et cible) et son méta-modèle (représentant la grammaire du langage de transformation) et produit en sortie le modèle cible. Le moteur de transformation peut procéder soit par interprétation ou par compilation. Le processus de transformation par modélisation est présenté sur la figure 3.7.

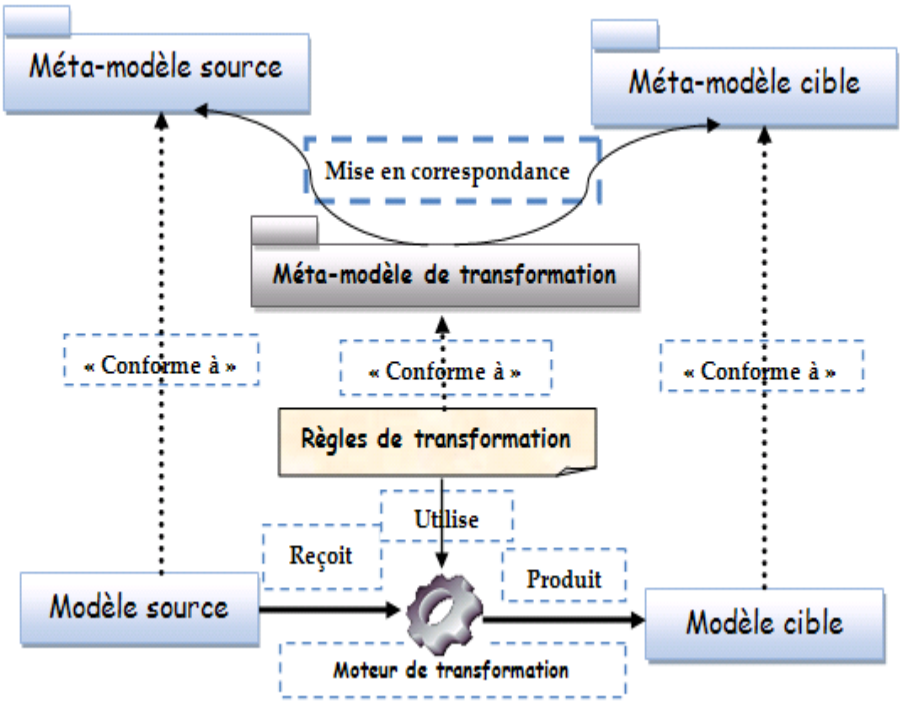


Figure 3.7 : Architecture d'un système de transformation basée sur la modélisation (OMG, 2003a)

3.4.2.2 Outils de transformations

Les transformations de modèles peuvent être manuelles, assistées par des outils ou entièrement automatiques. De nombreux outils, tant commerciaux que dans le monde de l'open-source sont aujourd'hui disponibles pour faire de la transformation de modèles. On peut grossièrement distinguer quatre catégories d'outils (Muller, 2006).

- **Les outils de transformation génériques**

Dans cette première catégorie, on trouve notamment d'une part les outils de la famille XML, comme XSLT ou Xquery, et d'autre part les outils de transformation de graphes (la plupart du temps issus du monde académique).

- **Les outils intégrés dans les ateliers de génie logiciel (AGL)**

Dans cette seconde catégorie, on trouve une famille d'outils de transformation de modèles proposée par des vendeurs d'AGLs, à l'instar de l'AGL Objecteering et plus particulièrement dans le monde le l'open source de Fujaba[16] (*From Uml to Java and Back Again*).

- **Les langages spécifiques (dédiés)**

Dans la troisième catégorie, on trouve des outils conçus spécifiquement pour faire de la transformation de modèles et prévus pour être plus ou moins intégrables dans les environnements de développement standard. Dans le monde académique, on trouve de nombreux projets open source s'inscrivant dans cette approche comme des outils ATL[17] (*Atlas Transformation Language*), Xtend[18] facilement intégrable dans la plateforme Eclipse comme des plugins que nous avons utilisés.

- **Outils de méta modélisation**

La dernière catégorie d'outils de transformation de modèles est celle des outils de méta-modélisation dans lesquels la transformation de modèles est une exécution d'un méta-programme.

3.5 Semantics of Business Vocabulary and Business Rules

Dans un contexte métier, l'OMG a publié en Janvier 2008 la première version[19] du standard SBVR qui définit un méta-modèle pour l'élaboration des modèles sémantiques du vocabulaire et des règles métier (OMG, 2008a). Ce standard est à destination des experts métier indépendamment de la modélisation des systèmes d'information. L'objectif premier est de leur fournir une base sémantique pour la construction de leur vocabulaire et règles métier favorisant ainsi leur échange entre les organisations (Chapin, 2005). Ainsi, le standard SBVR fournit une base formelle pour les règles métier et de ce fait, pour l'approche par règles métier

[16] FUJABA - http://www.fujaba.de

[17] ATL - http://www.eclipse.org/atl

[18] Xtend - http://www.eclipse.org/xtend

[19] La version 1.2 a été publiée en novembre 2013.

que nous avons présenté au Chapitre 2. Cette spécification a été une première étape dans le processus de standardisation des outils de gestion des règles métier (BRMS) existants. Elle constitue le premier standard dans la pile des standards de l'OMG qui fournit explicitement un modèle de la logique formelle (Nicolae et Wagner, 2008). SBVR fait entièrement partie de la couche de modèle métier du MDA comme le montre la figure 3.8.

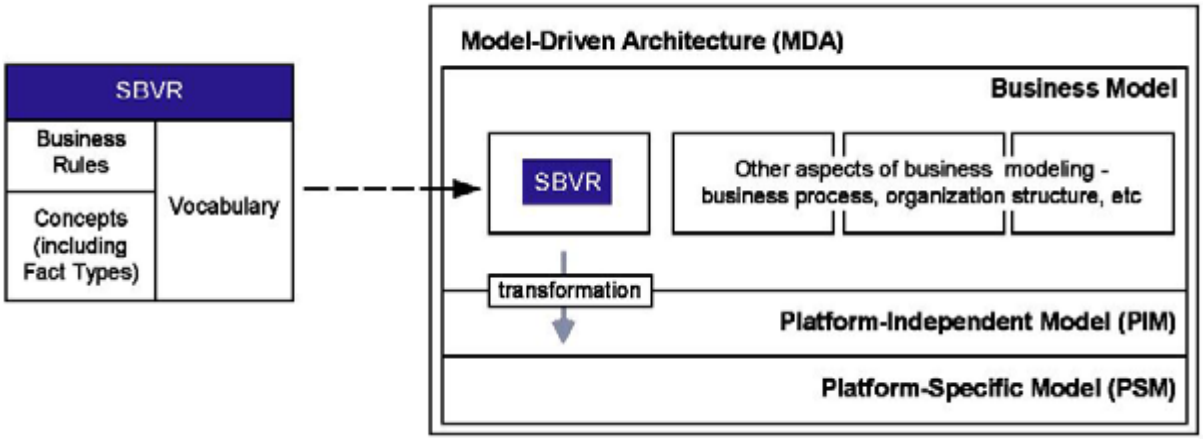


Figure 3.8 : Position du SBVR dans le MDA (OMG, 2008a)

Cette position implique deux remarques. La première est que SBVR s'adresse aux vocabulaires et règles métier des modèles métier. Les autres aspects des modèles métier doivent aussi être pris en compte et développés. Ces derniers incluent les processus métier et la structure organisationnelle, mais ceux-ci sont visés par d'autres standards au niveau de l'OMG. La deuxième remarque est que les modèles métier supportés par le SBVR ne s'adressent pas aux experts système mais aux experts métier, bien que le standard souligne la nécessité de pouvoir passer du CIM vers le PIM (Anderson et Spreeuwenberg, 2009). Le méta-modèle SBVR est donc conforme au standard MOF/XMI présenté plus haut et utilise la sérialisation XMI 2.1 pour l'échange des modèles.

Le méta-modèle SVBR n'est pas destiné à être utilisé par les experts métier directement car il ne s'agit pas des expressions ou des représentations de sens, mais plutôt il doit être utilisé pour décrire la structure sémantique formelle du langage des experts métier. Il n'est pas non plus destiné pour la conversation entre les experts métier, mais pour discuter des structures sémantiques sous-jacentes des concepts et des propositions issues des communications de l'entreprise (cf. figure 3.9). Par exemple, un expert métier n'a pas tendance à parler des

quantifications, mais il exprime d'une façon ou d'une autre les quantifications dans presque toutes les déclarations qu'il fait. Il n'a pas tendance à parler des conjonctions, des disjonctions, des négations logiques, des implications, etc. mais ces dernières font partie de la formulation de sa pensée. Ainsi, le méta-modèle SBVR que nous présentons à la section suivante consiste à décrire ces éléments conceptuels de sens utilisés dans la langue courante.

3.5.1 Le méta-modèle SBVR

Le méta-modèle SBVR est très complexe et sophistiqué (Linehan, 2008 ; Klener, 2009 ; Ruth, 2009). Les figures 3.9 et 3.10 donnent un aperçu du méta-modèle de l'approche où SBVR sépare la formulation sémantique et les concepts (*Meanings*).

3.5.1.1 Les concepts de SBVR

Il y a deux concepts (élément de sens) dans SBVR : le vocabulaire métier et les règles métier (OMG, 2008a).

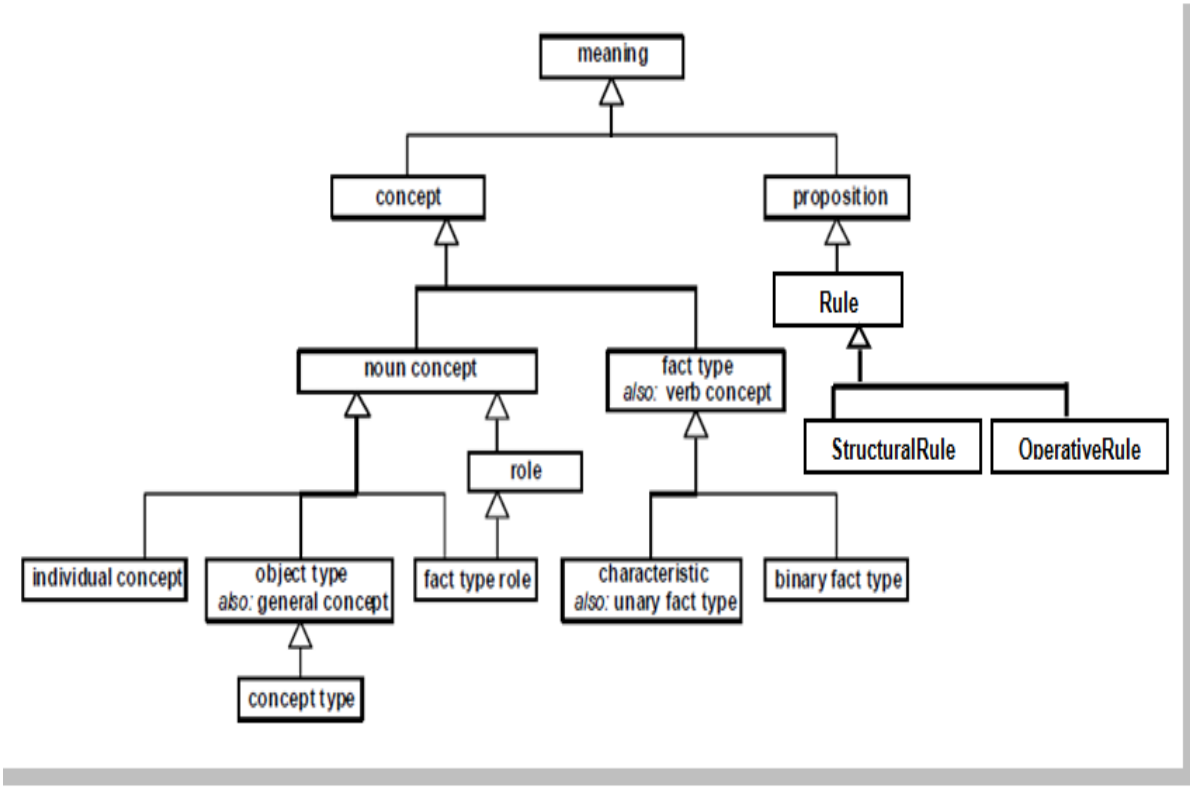


Figure 3.9 : Extrait du méta-modèle SBVR : *Meanings* (OMG, 2008a)

a. Vocabulaire métier

Le vocabulaire métier désigne un ensemble cohérent de concepts interconnectés qu'une organisation ou une communauté donnée utilise dans leur langage en vue d'effectuer leur métier. Ces concepts sont des entités qui sont représentées par des noms (*noun concepts*) et des faits qui sont des relations entre ces concepts (*verb concepts* ou *fact types*). Par exemple, une instance de *noun concept* peut être représentée ou désignée par un nom comme *Compte bancaire* ou *Client*. Une instance de *fact type* peut être représentée par *Client possède Compte bancaire.*

b. Règles métier

Dans SBVR, une règle métier est une règle sous la juridiction du métier qui définit la structure de l'entreprise et fournit les éléments de contrôle sur les actions. Il faut noter que la notion de règle métier ici se base sur les travaux du BRG car initialement SBVR a été développé par le BRG (OMG, 2004) qui a travaillé exclusivement dans ce domaine depuis la fin des années 1980. Ainsi, les notions clés dans SBVR sont présentées succinctement dans le manifeste du BRG (BRG, 2003) qui est présenté en Annexe I.

SBVR subdivise les règles métier en deux catégories :

- Les règles structurelles (*structural rule* ou *definitional rule*) : ce sont des règles qui portent sur la structure de l'entreprise ainsi que les éléments qui la composent. Elles expriment une nécessité, une possibilité ou une impossibilité.

  Exemple : il est nécessaire que chaque client possède au moins un compte bancaire.

- Les règles opérationnelles (*operational rule* ou *behavioral rule*) : ce sont des règles qui contrôlent la manière dont les activités du métier sont conduites. Elles sont destinées à des personnes et sont donc actionnables, mais pas nécessairement automatisables. Contrairement aux règles structurelles, les règles opérationnelles sont celles qui peuvent être directement violées par les personnes impliquées dans les activités du métier. Ainsi elles ont besoin de contraintes pour leur mise en application. Cependant ces contraintes de mise en application sont en dehors de la portée de SBVR. Les règles opérationnelles expriment une obligation, une interdiction ou une permission.

Exemple : Il est obligatoire qu‟un client retire au plus 300 euros dans un guichet automatique par jour.

3.5.1.2 Formulation sémantique (*Semantic Formulation*)

Comme nous avons déjà mentionné plus haut, la majeure partie du méta-modèle SBVR se focalise sur la formulation sémantique du vocabulaire et des règles métier qui est une caractéristique unique de SBVR. C‟est un ensemble de constructions qui structure le sens des règles ou la définition des concepts exprimée en langage naturel. Il s‟agit d‟une syntaxe abstraite et indépendante de tout langage pour représenter le sens d'une règle dans un ensemble de structures logiques de telle sorte que la machine puisse le traiter. On distingue deux types de formulations sémantiques comme le montre la figure 3.10 : les formulations logiques (*Logical formulations*) et les projections.

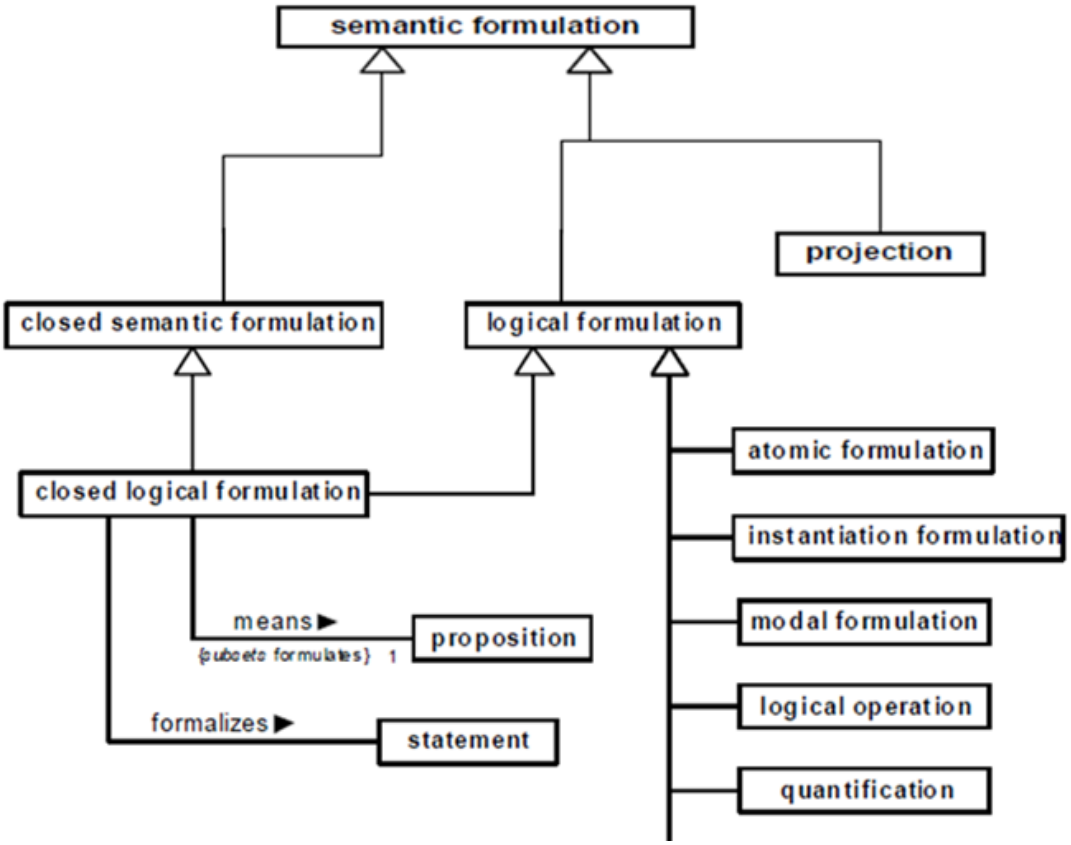


Figure 3.10 : Extrait du méta-modèle SBVR : *Semantic formulation* (OMG, 2008a)

a. Formulation logique

La formulation logique signifie les assertions (déclarations) logiques dans le langage de la logique formelle dont l'objectif est de formaliser les déclarations des experts métier. Elle structure les propositions à la fois simples et complexes. La logique formelle est la logique des prédicats du premier ordre (bien qu'il n'y ait aucune restriction sur l'extension en logique d'ordre supérieur), avec quelques extensions limitées dans la logique modale - notamment certaines formes déontiques, pour exprimer l'obligation et l'interdiction, et les formes aléthiques pour exprimer les nécessités.

Elle est par la suite spécialisée en : quantifications, opérations logiques, formulations atomiques basées sur les types de faits (*Fact types*), formulations d'instanciations, formulations modales, et d'autres formulations pour des fins spécifiques.

❖ Quantification

Une quantification est une formulation logique qui introduit une variable pour préciser la portée d'un concept. Elle peut être décomposée en sous catégories comme le montre la figure 3.11 (quantification universelle, existentielle, etc.).

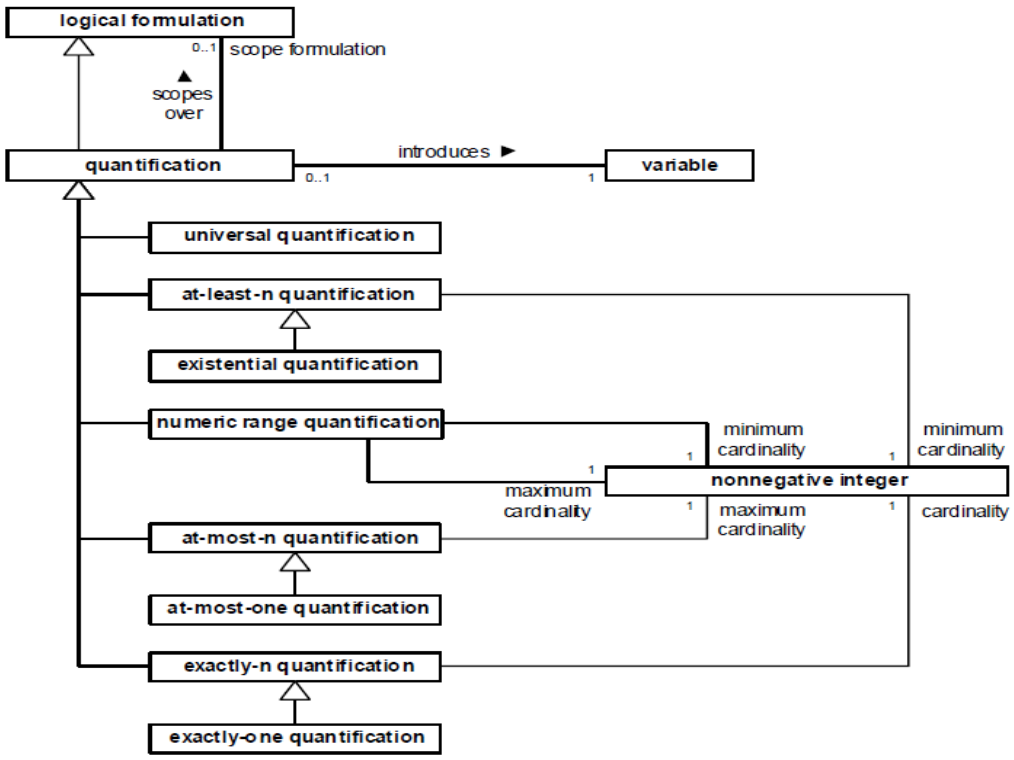


Figure 3.11 : Formulation logique : *quantifications* (OMG, 2008a)

❖ Opération logique

Une opération logique est une formulation logique qui formule un sens sur la base de vérité ou de non vérité du sens d'une ou plusieurs autres formulations logiques (ses opérandes logiques). Elle peut avoir deux opérandes (conjonction, disjonction, etc.) ou un opérande (négation). Elle est présentée sur la figure 3.12.

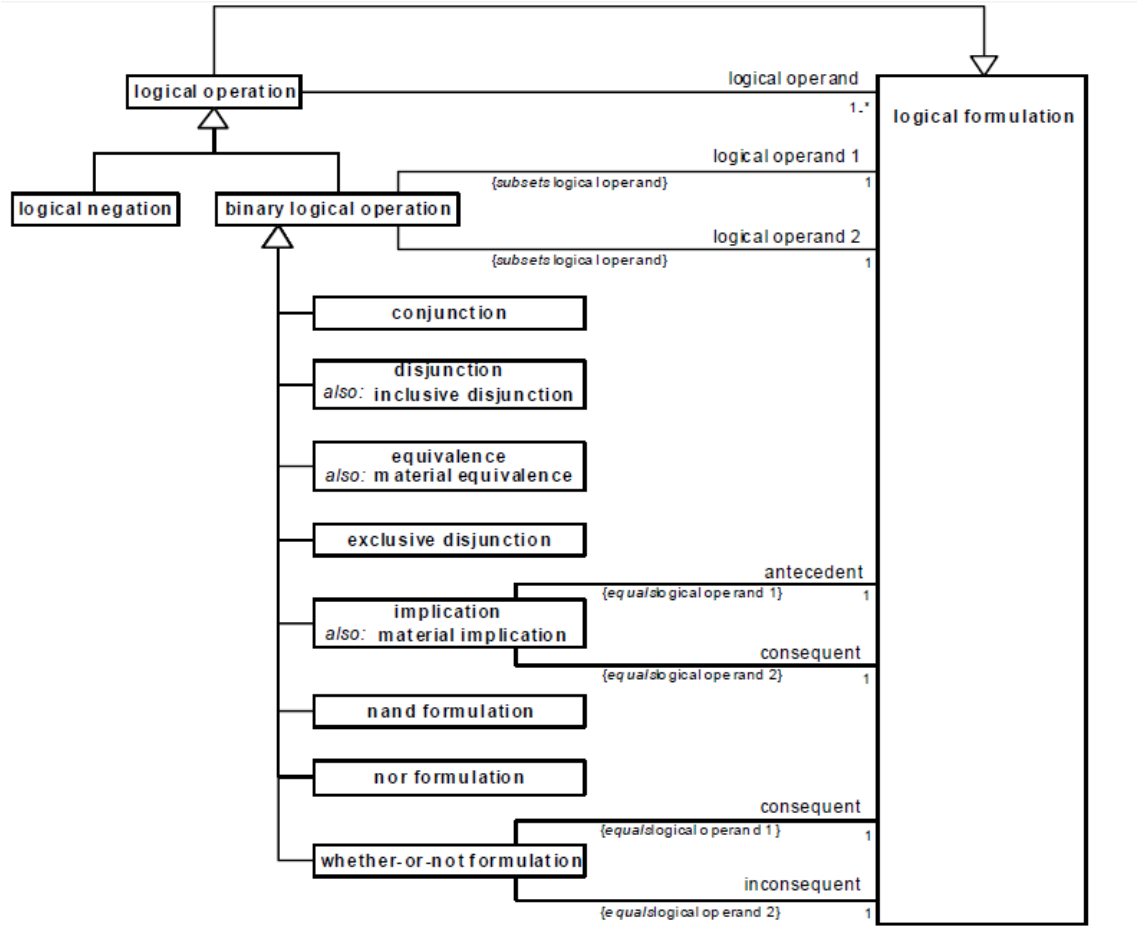


Figure 3.12 : Formulation logique : opérations logiques (OMG, 2008a)

❖ Formulation atomique

Une formulation atomique est une formulation logique qui formule le sens d'˜un type de fait (*Fact type*) avec des rôles dans une règle (cf. figure 3.13).

Exemple : customer has bank account.

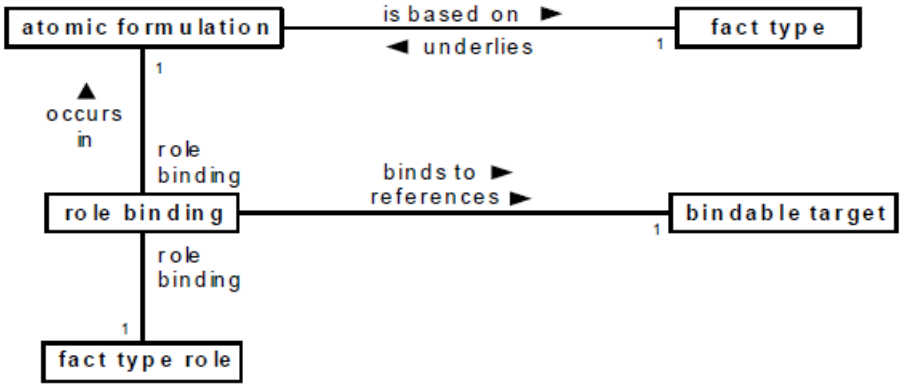


Figure 3.13: Formulation logique : formulation atomique (OMG, 2008a)

❖ Formulation d'instanciation

C'est une formulation logique qui formule l'instance d'un concept (cf. figure 3.14).

Exemple: « Silver account » is an instantiation of the noun concept « bank account ».

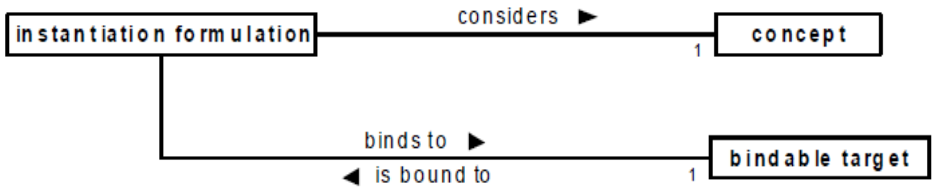


Figure 3.14 : Formulation logique : formulation d'instanciation (OMG, 2008a)

❖ Formulation modale

C'est une formulation logique qui formule le sens d'une autre formulation logique (cf. figure 3 .15).

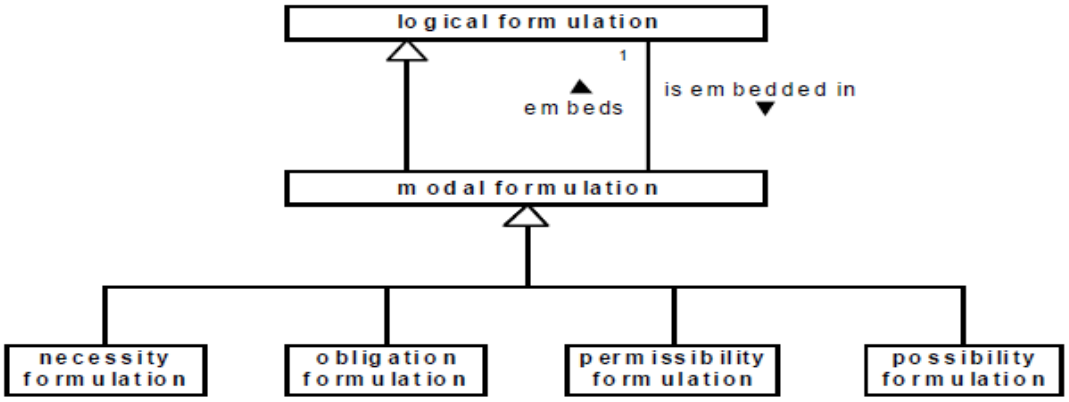


Figure 3.15 : Formulation logique : formulation modale (OMG, 2008a)

b. Les projections

Une projection est une formulation sémantique qui structure les intentions comme des ensembles de choses qui satisfont certaines contraintes et formulent des définitions, des agrégations et des questions. Elle est présentée sur la figure 3.16.

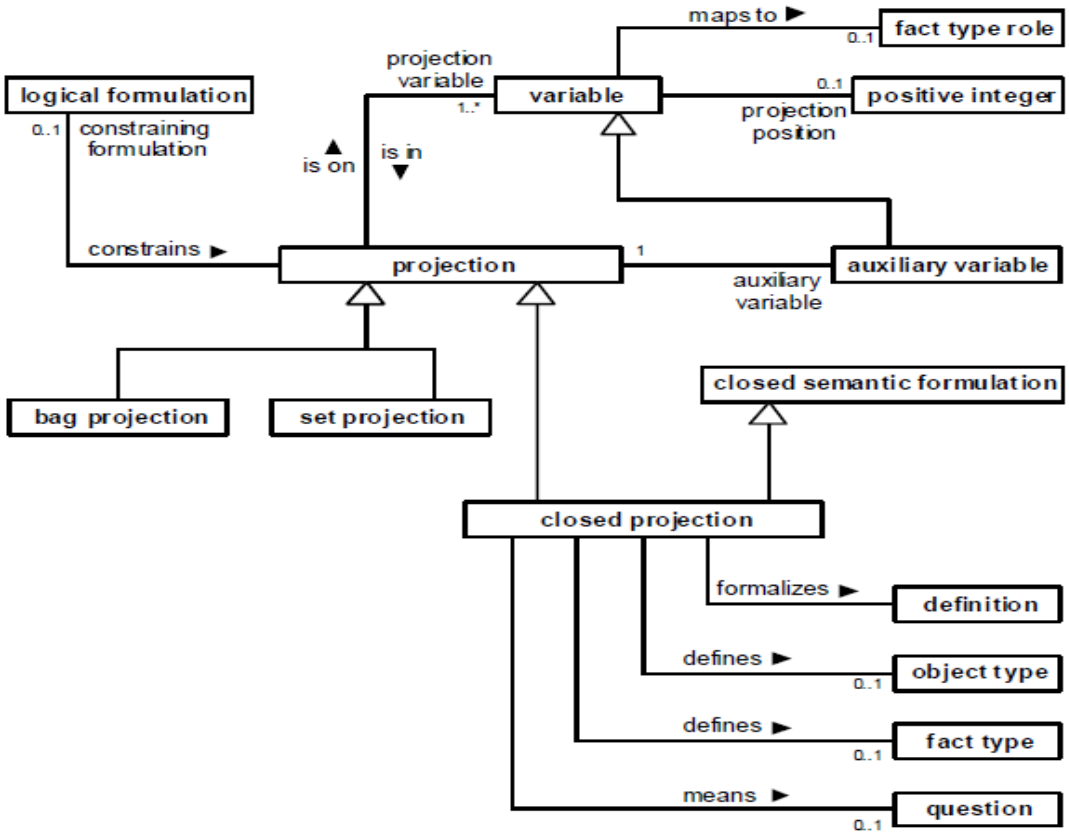


Figure 3.16: Projection (OMG, 2008a)

Les formulations sémantiques sont récursives. Plusieurs types de formulations sémantiques contiennent d'autres formulations sémantiques. Les variables logiques sont introduites par des quantifications et les projections de sorte que les formulations intégrées peuvent faire référence aux instances des concepts. Une variable logique utilisée dans une formulation est libre dans cette formulation si elle n'est pas introduite à l'intérieur de cette formulation. Une formulation est fermée si aucune variable n'est libre en son sein. Seule une formulation sémantique fermée (*closed semantic formulation*) peut formuler un sens. Si une formulation a une variable qui est libre, alors elle peut faire partie d'une plus grande formulation de sens (celle qui introduit la variable), mais elle ne formule pas elle-même un sens (OMG, 2008a).

Exemple : considérons les règles suivantes :

(1) *"It is necessary that each edited book has at least one editor"* (Ruth, 2009, p.116).

C'est une proposition basée sur une nécessité (*structural rule*), qui est sémantiquement structurée comme présentée sur la figure 3.17 :

```
The structural rule embeds a universal quantification.
. The universal quantification introduces a first variable.
. . The first variable ranges over the concept "edited book."
. The universal quantification scopes over an existential quantification.
. . The existential quantification has a minimum cardinality of 1.
. . The existential quantification introduces a second variable.
. . . The second variable ranges over the concept "editor."
. . The existential quantification scopes over an atomic formulation.
. . . The atomic formulation is based on the fact type "edited book has editor."
. . . . The atomic formulation has a role binding.
. . . . . The role binding is of the role "edited book" of the fact type.
. . . . . The role binding binds to the first variable.
. . . .The atomic formulation has a second role binding.
. . . . . The second role binding is of the role "editor" of the fact type.
. . . . . The second role binding binds to the second variable.
```

Figure 3.17 : Exemple de formulation sémantique de l'exemple (1)

(2) EU-Rent must appoint at least 3 officers. (OMG, 2008a, p.201).

C‟est une proposition basée sur une obligation (*Operational rule*), qui est sémantiquement structurée comme présentée sur la figure 3.18 :

```
The rule is a proposition meant by an obligation formulation
. That obligation formulation embeds an existential quantification
. . The existential quantification introduces a first variable
. . .  The first variable ranges over the individual concept ‘EU-Rent’
. . The existential quantification scopes over an at-least-n quantification
. . . The at-least-n quantification has the minimum cardinality 3
. . . The at-least-n quantification introduces a second variable.
. . . . The second variable ranges over the concept "officer"
. . . The at-least-n quantification scopes over an atomic formulation.
. . . .The atomic formulation is based on the fact type "company appoints officer"
. . . . . The atomic formulation has a role binding.
. . . . . .The role binding is of the role "company" of the fact type.
. . . . . .The role binding binds to the first variable.
. . . . . The atomic formulation has a second role binding.
. . . . . The second role binding is of the role "officer" of the fact type.
. . . . . .The second role binding binds to the second variable.
```

Figure 3.18 : Exemple de formulation sémantique de l‟exemple (2)

L'indentation dans les exemples de la figure 3.17 et de la figure 3.18 montre une structure hiérarchique dans laquelle une formulation sémantique à un niveau opère sur, ou quantifie sur, une ou plusieurs formulations sémantiques au niveau suivant. Chaque type de formulation logique, incluant la quantification et les opérations logiques, peut être incorporé dans une autre formulation logique à n'importe quel niveau de profondeur, et dans presque n'importe quelle combinaison. On note également que chaque ligne de l'exemple correspond à une instance d'un élément du méta-modèle SBVR.

Les figure 3.19 et figure 3.21 montrent respectivement les règles des exemples (1) et (2) comme des instances même du méta-modèle SBVR d‟un point de vue graphique. Les représentations complètes des règles comme des instances du méta-modèle SBVR sont assez

encombrantes et floues, comme le montre ces figures 3.19 et 3.21. Par conséquent, les versions de représentations des règles simplifiées, comme le montre les figures 3.20 et 3.22, sont les plus utilisées.

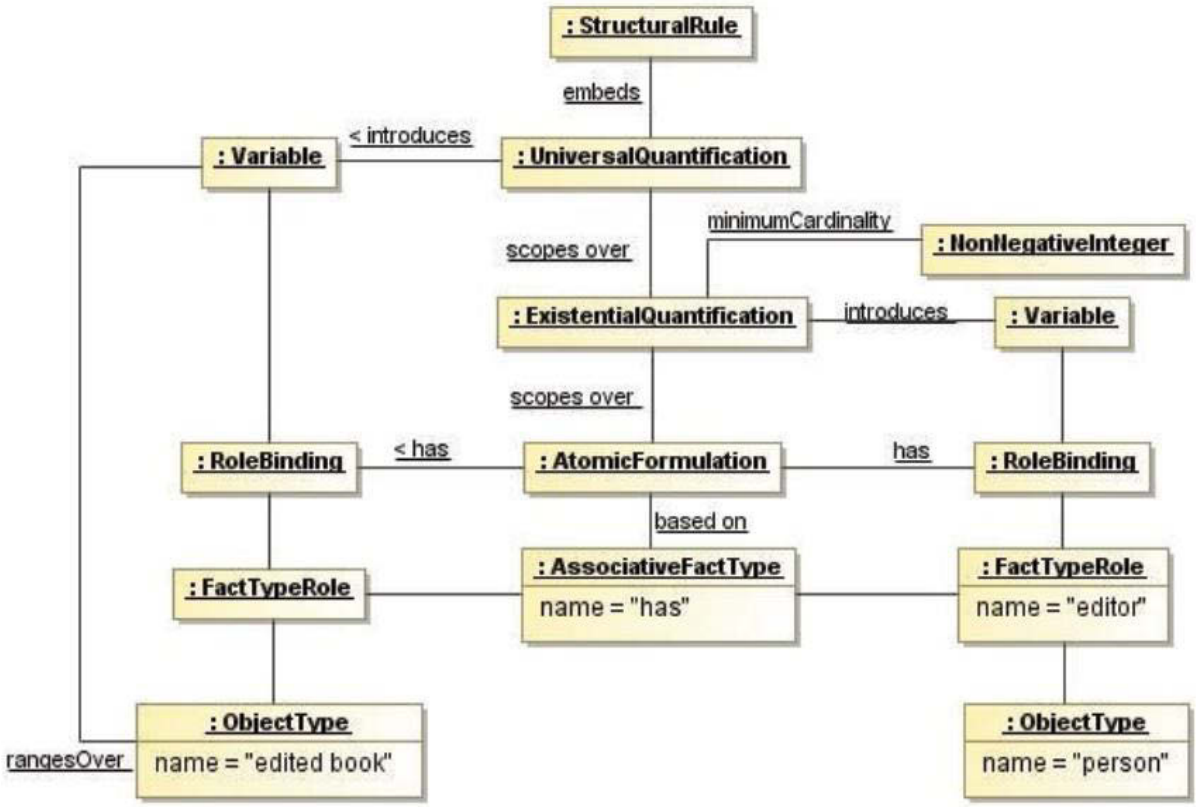


Figure 3.19 : Règle de l‟exemple (1) comme instance du méta-modèle SBVR

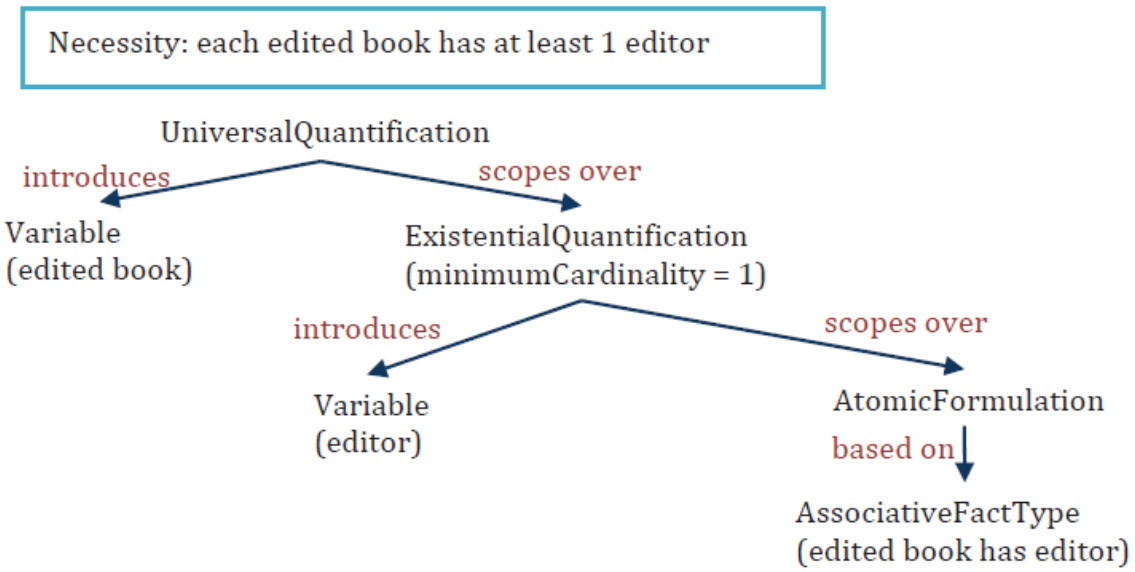


Figure 3.20: Version simplifiée de la règle de l‟exemple (1)

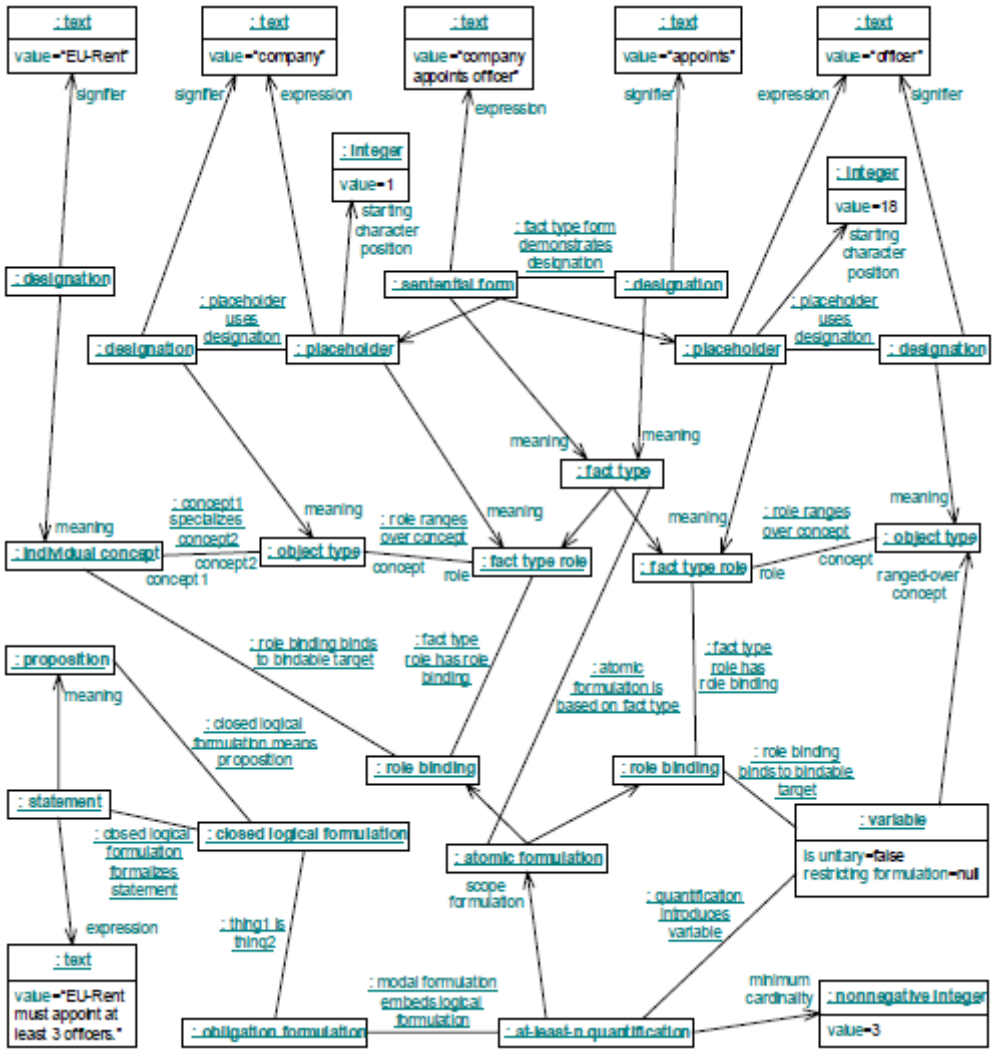


Figure 3.21 : Règle de l'exemple (2) comme instance du méta-modèle SBVR

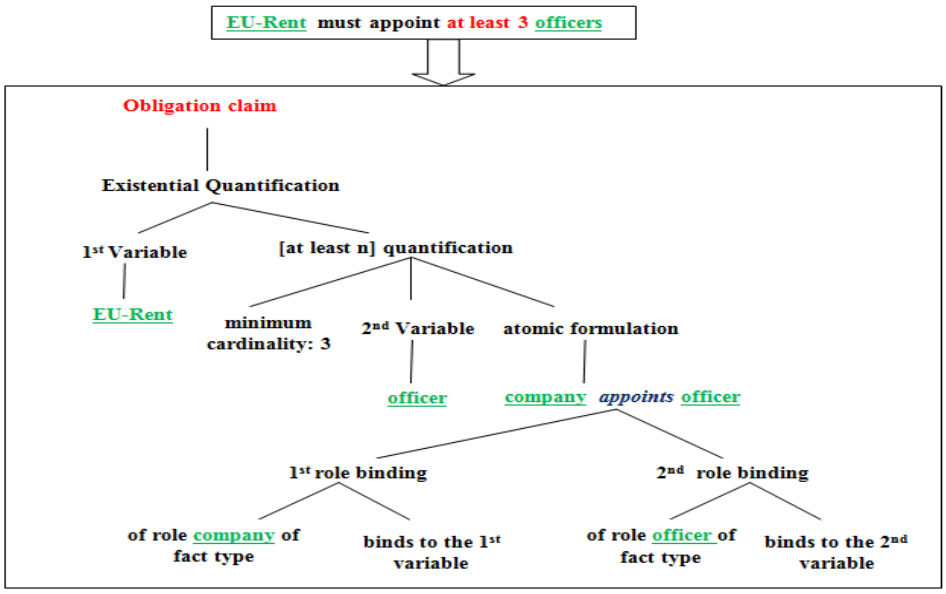


Figure 3.22 : Version simplifiée de la règle de l'exemple (2)

3.5.2 Notation ou représentation du vocabulaire et des règles métier

Les règles métier sont généralement exprimées en langage naturel, bien que certaines règles soient parfois illustrées graphiquement. SBVR ne fournit pas un langage logique pour ré-exprimer ces règles métier dans un autre langage que les experts métier n'utilisent pas. Plutôt, il fournit un moyen pour décrire la structure du sens des règles exprimées dans le langage naturel que les experts métier utilisent. Les formulations sémantiques présentées ci-dessus sont des structures logiques qui composent le sens. Ainsi, à cause des ambigüités du langage naturel, un langage naturel contrôlé ou structuré doit d"abord leur être proposé pour exprimer formellement leurs règles avant la transformation vers la structure sémantique SBVR.

Pour diverses raisons, le standard SBVR n"a pas défini une spécification normative de la syntaxe à utiliser pour l"expression du vocabulaire et des règles métier par les experts métier, en partie parce que certains groupes ou entreprises impliqués dans la spécification SBVR sont des fournisseurs d"outils et ont déjà des langages propriétaires dans leur outillage comme IBM et Blaze Advisor (Ruth, 2009).

SBVR Structured-English (OMG, 2008a) et *RuleSpeak*[20] sont deux notations définies dans la spécification SBVR à titre non normatif qui peuvent être mappées au méta-modèle SBVR. Cependant elles ne sont pas formelles et ne peuvent donc pas être traitées de manière entièrement automatique. La syntaxe est obtenue par une mise en forme manuelle avec des styles et des couleurs de polices de caractères facilitant ainsi la compréhension du lecteur humain.

Par exemple : Le *SBVR Structured-English* définit quatre styles et couleurs de mise en forme comme suit :

- Les *Noun Concept* sont représentés par des expressions en vert soulignées.
  Exemple : bank account, customer , order, etc.
- Les *Individual Concepts* sont représentés par des expressions en vert doublement soulignées. Exemple : France, Euro, etc.
- Les *Fact types* sont représentés par des expressions en bleu et en italique.
  Exemple : customer *places* order, etc.
- Les autres mots-clés sont en rouge.

[20] RuleSpeak - http://www.rulespeak.com/en

Exemple : each, exactly one, and, or, etc.

La spécification SBVR est elle même écrite en utilisant les styles et le formatage de *SBVR Structured-English*. En outre, en Juin 2008, l'OMG a présenté une demande pour une proposition (*Request for proposal*) visant à définir au moins un langage standard dans lequel les experts métier pourront exprimer leur vocabulaire et leurs règles métier (OMG, 2008c).

Au vu de ceci, la spécification SBVR est restreinte à la sémantique formelle du vocabulaire et des règles métier laissant ainsi de coté un constituant clé du langage qui est la syntaxe. Ceci constitue un enjeu crucial car l'analyse de la syntaxe du langage métier en vue de l'élaboration des modèles sémantiques SBVR est un gros problème. Cette thèse contribue ainsi dans cette direction et ceci par une montée en abstraction grâce à un langage contrôlé RuleCNL pour permettre aux experts métier de spécifier facilement leur vocabulaire, leurs règles métier et un moteur de transformation vers des structures sémantiques SBVR comme indiqué sur la figure 3.23.

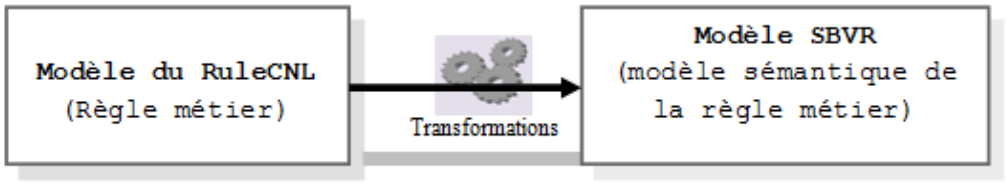


Figure 3.23 : Architecture de transformation d'un modèle RuleCNL en modèle SBVR

3.6 Conclusion

Dans ce chapitre, nous avons tout d'abord présenté l'approche MDA qui décrit le cadre normatif de l'IDM autour de ces concepts de base, son architecture et sa mise en œuvre. Ensuite, nous avons présenté les technologies clés du MDA que sont la méta-modélisation et les transformations de modèles qui sont les moyens/techniques que nous utilisons pour la mise en œuvre de notre langage contrôlé RuleCNL. Enfin, le standard SBVR a été décrit autour de son méta-modèle sémantique nécessaire pour la définition de la sémantique de RuleCNL. Dans le chapitre suivant, nous présentons un état de l'art sur les langages contrôlés en général et sur les langages contrôlés pour l'expression des règles métier en particulier.

Chapitre 4

État de l'art sur les langages contrôlés

4.1 Introduction

Dans les chapitres 2 et 3, nous avons respectivement présenté l'ARM et le MDA qui sont deux méthodologies parallèles et éprouvées dans la modélisation des systèmes d'information. Ces deux méthodologies sont complémentaires et doivent être couplées afin de faire face aux problèmes récurrents auxquels font face les entreprises de nos jours à savoir l'alignement de leur SI avec l'évolution des besoins du métier ou de l'entreprise. Toutes les deux méthodologies offrent des avantages réels à différents niveaux d'abstraction et sont toutes à la quête de l'agilité, avec des leviers permettant de séparer la logique système de la logique métier afin de s'adapter très rapidement et à moindre coût aux changements tant au niveau du métier que dans les SI. L'ARM préconise d'externaliser les règles métier des SI et permet leur gestion de façon centralisée. Le MDA à travers ses modèles abstraits en couche permet aux experts systèmes de modéliser ces règles métier à différents niveaux de leur cycle de vie afin de faciliter leur implémentation effective dans les SI. Cependant, pour un alignement effectif des SI avec les besoins métier, l'implication des experts du métier est une condition sine qua non. Ainsi, il est indispensable que la modélisation des règles métier commence par le point d'origine à savoir par les experts métier qui ne sont pas forcement des informaticiens, mais sont ceux qui connaissent mieux que quiconque les besoins du métier ainsi que les exigences des SI qui les automatisent et sont à la base des prises de décisions.

Le langage naturel des experts métier est probablement le langage le plus expressif pour la spécification des règles métier qui existe. Il est facile pour les humains de l'utiliser et de le comprendre. Cependant, ce pouvoir expressif rend le traitement du langage naturel difficile pour un ordinateur parce que beaucoup d'informations pertinentes sont généralement implicites et la plupart de temps déduites par les humains. Les langages formels de règles (Monin, 2003) proposés par les différents BRMS existants pour la spécification des règles métier ont une syntaxe et une sémantique bien définies, et permettent des raisonnements automatisés. Ces langages cependant sont souvent assez difficiles pour les spécialistes du

domaine. Ainsi, les langages contrôlés (LC) apparaissent comme un bon compromis pour interfacer le langage naturel des experts métier et le langage formel des BRMS à destination des machines. Les LCs sont des sous-ensembles des langages naturels qui sont limités d'une manière à permettre leur traitement automatique et leurs transformations automatiques vers des représentations formelles. Ils sont en général plus faciles à comprendre par l'homme et plus facile à traiter la machine (Schwitter, 2010).

4.2 Motivations pour les langages contrôlés dans les SI

«We must develop languages that the scientist, the architect, the teacher, and the layman can use without being computer experts. The language for each user must be as natural as possible to her/him. The statistician must talk to his terminal in the language of statistics. The civil engineer must use the language of civil engineering. When a man learns his profession he must learn the problem-oriented languages to go with that profession» (Martin, 1967).

« We must constantly turn to new languages in order to express our ideas more effectively. Establishing new languages is a powerful strategy for controlling complexity in engineering design; we can often enhance our ability to deal with a complex problem by adopting a new language that enables us to describe (and hence to think about) the problem in a different way, using primitives, means of combination, and means of abstraction that are particularly well suited to the problem at hand » (Abelson et Sussman, 1996).

Dans le contexte des systèmes d'information, environ 66% des grands projets échouent et conduit soit au dépassement du budget initial, soit au déploiement d'un système qui n'est jamais mis en production (Standish, 2004). Parmi les causes principales de cet échec, on trouve principalement le manque d'implication des experts métier et des exigences système mal spécifiés ou ambigües. Dans les approches traditionnelles de développement logiciel, les communications entre les experts métier et les ingénieurs logiciels se font en amont en utilisant le langage naturel qui peut être source d'incohérences et d'erreurs d'interprétations. Très souvent pendant le processus de développement, les ingénieurs logiciels s'attendent à ce que les experts métier apprennent leur langage technique de modélisation comme par exemple des diagrammes UML. Malheureusement, on se rend compte que cela ne fonctionne toujours pas et cela se traduit par un manque d'implication de ces experts métier. Face à cette situation, Il est clair que les choses doivent se faire différemment. En plus, dans le monde actuel, les

ordinateurs deviennent de plus en plus répandus, chaque entreprise ou chaque secteur d'activité doit faire face à des ordinateurs dans leur vie quotidienne. Toutefois le pourcentage de personnes ayant une formation supérieure en informatique reste très faible (Kuhn, 2010). En conséquence, de plus en plus de personnes n'ayant pas de formation en informatique ont besoin d'interagir avec ces ordinateurs. Ainsi la recherche actuelle se tourne vers des méthodes permettant aux experts métier d'exprimer directement leur connaissance de façon naturelle et formelle aux ingénieurs logiciels.

Cette situation soulève la nécessité de communiquer avec les ordinateurs d'une manière facile et intuitive qui ne nécessite pas de formation particulière de l'utilisateur. Cependant, le fait que les humains et les ordinateurs utilisent des langages de types complètement différents est un obstacle majeur pour la communication entre les deux. Les ordinateurs utilisent des langages formels (comme les langages de programmation et langages d'ontologie basés sur la logique) alors que les humains s'expriment en langage naturel (comme l'anglais, le français, etc.). La solution la plus facile pour ce problème est d'écrire des programmes informatiques qui sont en mesure de traiter le langage naturel d'une manière fiable. Ceci renvoie aux domaines de la recherche en TAL qui s'est avéré être un problème extrêmement difficile. Une grande quantité de recherches ont été consacrées à ce problème au cours des décennies et malgré quelques progrès notables, les ordinateurs ne parviennent toujours pas à traiter le langage naturel d'une manière générale et fiable. Toutefois, beaucoup de directions ont été prises pour le TAL par certains auteurs. Parmi les plus remarquables, nous pouvons citer l'analyse linguistique micro-systémique et le traitement automatique des langues assisté par ordinateur.

4.2.1 L'analyse linguistique micro-systémique

L'analyse linguistique micro-systémique ou MSLA (*Micro-systemic Linguistic Analysis*) (Cardey, 1987 ; 2013) est fondée sur le postulat que le langage peut être segmenté en des systèmes individuels sur la base de l'influence que ces systèmes exercent l'un sur l'autre. L'analyse linguistique de tous les niveaux du langage peut être décrite de cette manière (lexique, syntaxe, morphosyntaxe, sémantique, morphosémantique et d'autres). La MSLA propose que pour être traités de façon efficace, les langages naturels doivent être décomposés en des systèmes qui peuvent être analysés à la fois par un être humain et par la machine car ils sont assez petits, mais complets et sont ainsi en mesure de fusionner ensemble comme un

système unifié. En plus de cela, les systèmes ainsi délimités peuvent interagir avec d'autres systèmes, et cette interaction est une propriété du langage car rien n'est indépendant ; le lexique, la morphologie et la syntaxe sont liés.

La méthodologie consiste donc à décomposer l'analyse des systèmes linguistiques en des systèmes à composants comme suit (Cardey et Greenfield, 2005) :

Sv : représentant le système des variants ;

Sc : représentant le système canonique ;

Ss : représentant le super-système qui met les deux systèmes Sc et Sv en relation l'un avec l'autre.

La MSLA possède des applications variées (Cardey et Greenfield, 2006). Elle a été utilisée avec succès pour plusieurs applications concrètes comme la désambigüisation de l'étiquetage des parties du discours (Cardey et Greenfield, 2003), les langages contrôlés et la traduction automatiques dans le domaine de la sécurité en général (Cardey et al., 2008 ; Cardey, 2011), dans la normalisation, la ré-accentuation automatique des textes en français (Feuto, 2011 ; 2012), etc.

Le MSLA trouve ses bases dans la modélisation mathématique, elle comporte des propriétés inhérentes qui satisfont certaines notions pertinentes du génie logiciel et de plus, ses fondations mathématiques apportent les méta-descriptions dont a besoin le processus de développement du logiciel (Cardey et Greenfield, 2008).

4.2.2 Le traitement automatique des langues assisté par ordinateur

Mitkov (Mitkov, 2005) propose plutôt le traitement automatique des langues assisté par ordinateur ou CANLP (*Computer-Aided Natural Language Processing*). Dans cette approche, le traitement du langage ne sera pas fait entièrement par les machines, mais doit nécessiter une intervention humaine qui doit améliorer ou valider les résultats de sortie des ordinateurs. Plusieurs systèmes basés sur l'approche CANLP existent dans différents domaines du TAL :

- *Machine-Aided Translation*
- *Computer-Aided Text Summarisation*

- *Computer-Aided Generation of Multiple-choice texts*
- *Computer-Aided Information Extraction*
- *Annotation of corpora*
- *Computer-Aided text simplification* (Temnikova, 2012).

Cependant, les statistiques montrent que le TAL rencontre toujours d'énormes problèmes et de nos jours, les ordinateurs sont toujours incapables de traiter le langage naturel de façon fiable. Les ordinateurs ne parviennent pas à comprendre le langage naturel et les humains ont du mal à apprendre les langages formels ou de programmation (Jansen et al., 1998 ; Rector et al., 2004). On peut donc conclure que les humains et les ordinateurs doivent communiquer, mais sans toutefois parler le langage de l'autre. Dans certains cas, l'utilisation des diagrammes graphiques peut résoudre ce problème jusqu'à un certain degré. Les schémas conçus correctement peuvent être compris intuitivement sans aucune formation particulière. À un certain point cependant, quand il s'agit de la représentation de situations complexes, cela ne fonctionne plus. Les schémas graphiques perdent leur intelligibilité intuitive s'ils deviennent trop complexes.

Ainsi, la solution la plus prometteuse est portée sur les langages contrôlés afin de faire face aux ambigüités du langage naturel qui reste un problème inhérent au traitement automatique des langues. Les langages contrôlés, étant un sous domaine du TAL, ont été proposés pour permettre aux humains d'exprimer leurs connaissances d'une manière naturelle mais pouvant aussi être comprises par les ordinateurs. Ils sont des sous-ensembles artificiels du langage naturel (anglais, français, etc.) dans le but d'améliorer la communication entre les humains et les ordinateurs. D'une part, un LC ressemble à un langage naturel et est intuitivement compris par les locuteurs natifs du langage naturel sous-jacent, d'autre part, il est limité de manière à ce que les ordinateurs puissent le traiter et l'interpréter. L'idée est de définir des langages qui partagent l'intelligibilité intuitive humaine du langage naturel avec la capacité de leur traitement automatique sur la base de la logique formelle.

L'objectif poursuivi dans cette thèse est de contribuer dans cette direction en proposant un langage contrôlé pour la spécification des règles métier qui est défini de façon formelle, et de ce fait compréhensible pour les ordinateurs, mais ressemble au langage naturel. L'idée sous-jacente est que les règles métier exprimées dans un langage formel ou de programmation

peuvent toujours être exprimées d'une manière assez naturelle. Cette thèse porte sur l'anglais et le français comme langages naturels sous-jacents.

Dans les sections suivantes, nous présentons une vue d'ensemble sur les langages contrôlés. Nous commençons par présenter leurs origines, leurs définitions, typologies ainsi que les langages contrôlés existants pour les règles métier.

4.3 Vue d'ensemble sur les langages contrôlés

4.3.1 Origine des langages contrôlés

Au 20e siècle, plusieurs sous-ensembles contrôlés des langages naturels principalement l'anglais ont été définis dans le but d'améliorer la communication entre les humains dans le monde entier. Un des premiers d'entre eux était le *Basic English* (Ogden, 1930) qui a été présenté en 1930 par Charles Ogden. Il restreint la grammaire par un ensemble de règles et possède un vocabulaire de seulement 850 mots anglais. L'objectif premier était l'enseignement de l'anglais aux locuteurs non natifs. Il a par la suite été étendu pour servir de base commune pour la communication en politique, en économie et science (Ogden, 1932). D'autres exemples comprennent le *Special English* qui est un anglais simplifié utilisé depuis 1959 pour l'actualité dans les medias (radio et télévision) et encore présenté quotidiennement par le service de radiodiffusion aux États-Unis the *Voice of America* (VOA) (Voice of America, 2009). *Seaspeak* (Strevens et Johnson, 1983) est aussi un langage contrôlé international définit en 1983 pour la communication entre les navires et les ports. *PoliceSpeak* (Johnson, 2000) est également un LC qui a été développé pour améliorer la communication entre les officiers de Police français et anglais dans le Tunnel. Le projet a été lancé en 1988 et financé par le British Telecom, le Home Office et le comté de Kent, et le langage a été prêt en 1992.

À partir des années 1970, des sous-ensembles contrôlés de l'anglais ont été développés essentiellement pour des domaines techniques et principalement pour la rédaction de la documentation technique. Des entreprises comme Caterpillar (Verbeke, 1973), Boeing (Hoard et al., 1992), IBM (Bernth, 1997), Xerox (Ruffino, 1982), General Motors (Means et Godden 1996 ; Means et al., 2000), et bien d'autres ont défini leurs propres sous-ensembles de l'anglais qu'elles ont utilisés pour leurs documentations techniques. L'objectif était de réduire l'ambiguïté de leurs documents et d'éviter les incompréhensions, surtout pour les locuteurs

non anglophones. Cet effort était motivé par la complexité croissante de leur documentation technique et leur activité accrue sur des frontières linguistiques. Ces langages contrôlés sont encore utilisés de nos jours. Par exemple, le plus célèbre est l‟ASD-STE 100[21] (*Aerospace and Defense Industries Association of Europe Simplified Technical English*) anciennement connu sous le nom de AECMA (Association Européenne des Constructeurs de Matériel Aérospatial) (AECMA, 1986), qui est largement utilisé dans le domaine de l'aéronautique. Il a été mis en place afin d'éviter les accidents mortels dus à des malentendus. Cependant, ces langages contrôlés sont conçus pour améliorer seulement la communication entre les humains, ils n'ont pas de sémantique formelle et ne mettent pas l'accent sur leur traitement automatique (Kuhn, 2010).

Toujours dans les années 1970, les interfaces en langage naturel pour les bases de données sont devenues populaires (Hendrix et al., 1978 ; Waltz, 1978), ce qui a également conduit à l'étude des langages contrôlés dans ce contexte (Mueckstein, 1985). Sans doute, en raison des problèmes insurmontables du TAL, la recherche s'est éloignée encore de ce sujet (Kuhn, 2010).

Le langage de programmation COBOL mérite bien d‟être mentionné ici, même s‟il ne peut pas être considéré comme un langage contrôlé au sens strict mais considéré par (Sowa, 2000) comme un langage contrôlé. Émergent à la fin des années 1950, il est l'un des plus anciens langages de programmation, mais encore l'un des plus largement utilisés aujourd'hui (Mitchell, 2006). COBOL est orienté vers les systèmes d‟entreprises et d'administrations et utilise un grand nombre de mots-clés qui rend le code COBOL semblable au langage naturel et de ce fait plus facile à comprendre.

Néanmoins, bien que le langage COBOL utilise des phrases naturelles où d'autres langages de programmation utilisent des symboles ou des mots-clés courts, la structure de déclaration ne suit pas vraiment une grammaire naturelle et les programmes en COBOL semblent toujours très artificiels et ne peuvent être compris sans formation.

Même si l'approche générale a déjà été proposée dans les années 1970 (Skuce, 1975), ce n‟est que dans le milieu des années 1990, lorsque les langages comme ACE (*Attempto Controlled English*), PENG (*Processable English*), ont été réellement implémentés afin de contrôler les

[21] http://www.asd-ste100.org

langages naturels de sorte qu'ils puissent être mappés directement vers des logiques formelles. Cela peut être considéré comme une revitalisation de l'esprit des premiers temps de la logique où les expressions logiques étaient exprimées en langage naturel. Le but est de rendre à nouveau la logique compréhensible pour des personnes sans formation particulière en logique.

Ainsi, les langages contrôlés ont été utilisés dans les domaines et applications scientifiques variés. Ils ont été utilisés pour des preuves mathématiques (Cramer et al. 2010), dans le domaine médical pour rédiger des protocoles de messages (Renahy et al., 2012 ; Renahy, 2009 ; Vuitton et al., 2009 ; Renahy et al., 2010), dans le domaine du web sémantique pour des interfaces en langage naturel qui peuvent être transformer en OWL (*Ontology Web Langage*) *Rabbit* (Hart, Johnson et Dolbear, 2008), *CLOnE* (Funk et al., 2007), *Lite Natural Language* (Bernardi et al., 2007), *Manchester OWL Syntax* (Horridge et al. 2006), dans le domaine de la sécurité globale (Cardey et al., 2010 ; Renahy et al., 2009) et l'aéronautique (Spaggiari et al., 2003), pour la traduction automatique (Wu et al., 2004 ; Khruathong, 2004 ; Cardey, 2005 ; Cardey et al., 2005 ; Cardey et al., 2008 ; Cardey, 2011), pour la simplification des documents (Temnikova, 2012), dans des mémoires de corrections d'erreurs pour les documentations techniques (Kang et Saint-Dizier, 2014), etc. Bien que les LCs soient principalement développés pour l'anglais, il existe aussi des exemples pour d'autres langages naturels tels que le français (Barthe, 1996, 1998 ; Cardey, 2011 ; Renahy et al, 2010), l'allemand (Schactl, 1996), le Suédois (Alqvist et al., 1996), l'espagnol (Bustamante, 2000), (Blanco 2009), le japonais (Isahara, 2013), le chinois (Zhang, 1998), le mandarin (Pool, 2006), le grec moderne (Vassiliou et al, 2003), le polonais (Bogacki, 2009 ; Rudas 2009), le thaï (Thongglin, 2014) avec des prototypes existants aussi pour le bulgare (Temnikova et Margova, 2009).

4.3.2 Définition des langages contrôlés

Le terme « *langage contrôlé »* ou « *langage naturel contrôlé »* utilisé dans cette thèse a connu beaucoup de dénominations durant ces quatre dernières décennies (*Controlled natural language* , *Processable language* , *Simplified language* , *Technical language* , *Structured language* , *Basic language* , *Domain Specific language,* etc.). Des variétés de tels langages ont été conçues pour des applications et des domaines différents comme nous l'avons présenté dans la section précédente. Il y a deux raisons principales à cela (Kuhn, 2014) :

Tout d'abord pendant plusieurs décennies (à partir des années 1930 jusqu'à nos jours), les langages contrôlés ont non seulement émergé d‟environnements différents (industrie, universités, gouvernement), mais aussi de disciplines différentes (informatique, philosophie, linguistique, et ingénierie). Les utilisateurs avec des profils divers ont souvent utilisé et continuent d'utiliser des noms différents pour désigner ce même type de langage.

Deuxièmement, les langages contrôlés présentent également quelques différences apparentes : certains sont intrinsèquement ambigus, d'autres sont aussi précis que la logique formelle ; pratiquement tout peut être exprimé dans certains cas, et seulement très peu dans d'autres ; certains semblent tout à fait naturels, d'autres ressemblent plus à des langages de programmation informatique ; certains sont définis par un ensemble restreint de règles de grammaire, d'autres sont si complexes qu'aucune grammaire complète existe. Toutefois, malgré ces diversités, ces langages contrôlés partagent des propriétés importantes. Ils ont été en général utilisés pour améliorer la communication entre les humains, améliorer la qualité de la traduction automatique, fournir des interfaces et des représentations intuitives en langage naturel pour des notations formelles.

Pour ces raisons, il existe plusieurs définitions dans la littérature. Les deux citations suivantes l‟illustrent :

"*A controlled language is a restricted version of a natural language which has been engineered to meet a special purpose, most often that of writing technical documentation for non-native speakers of the document language. A typical CL uses a well-defined subset of a language's grammar and lexicon, but adds the terminology needed in a technical domain*." (Kittredge, 2003).

"Controlled natural language is a subset of natural language that can be accurately and efficiently processed by a computer, but is expressive enough to allow natural usage by non-specialists." (Fuchs et Schwitter, 1995).

Ces deux citations présentent une forte tendance pour un type particulier de langage contrôlé (que nous allons discuter plus en détail ci-dessous) : la première citation se concentre sur les langages techniques qui sont conçus pour améliorer la compréhension, tandis que la seconde ne porte que sur les langages qui peuvent être interprétés par des ordinateurs. Elles sont d'accord, cependant, sur le fait qu'un LC est basé sur un certain langage naturel mais est plus

restrictif. Il est généralement admis que les LCs sont des langages construits, c‟est-à-dire pas des langages qui ont émergés naturellement (*sublanguages*), mais ont été conçus.

L'utilisation du terme sous-ensemble peut s‟avérer mathématiquement trompeuse car, de nombreuses LCs ne sont pas des sous-ensembles propres du langage naturel sous-jacent. Beaucoup de ces langages peuvent avoir de petits écarts par rapport à la grammaire ou la sémantique du langage naturel. D'autres font appel à des éléments non naturels tels que les couleurs et les parenthèses pour augmenter la lisibilité et la précision. La relation de sous-ensemble dans son sens mathématique est clairement trop stricte pour couvrir une grande partie des langages communément appelé langages contrôlés. Bien qu'ils partagent tous clairement des propriétés importantes, les langages spécifiques peuvent être très différents dans leur couverture et par leur nature.

Pour répondre à ces problèmes, un langage est appelé un langage naturel contrôlé si et seulement s‟il remplit toutes les quatre propriétés suivantes (Kuhn, 2014) :

1. Il est basé exactement sur un langage naturel (son langage de base).

2. La différence la plus importante entre lui et son langage de base (mais pas nécessairement la seule) est qu'il a un lexique, une syntaxe et/ou une sémantique restreints.

3. Il conserve la plupart des propriétés naturelles de son langage de base, de sorte que les locuteurs de son langage de base peuvent intuitivement et correctement comprendre des textes dans ce langage naturel contrôlé, au moins à un degré substantiel.

4. Il est un langage construit, ce qui signifie qu'il est explicitement et consciemment défini, et n'est pas le produit d'un processus implicite et naturel (même s‟il est basé sur un langage naturel qui est le produit d'un processus implicite et naturel).

Au vue de ces caractéristiques, nous pouvons définir un langage contrôlé comme :

“A controlled natural language is a constructed language that is based on a certain natural language, being more restrictive concerning lexicon, syntax and/or semantics while preserving most of its natural properties” (Kuhn, 2014).

Il convient de noter que le terme « langage » est utilisé ici dans un sens qui est limité aux langages textuels et exclut des langages visuels tels que les diagrammes et autres. La

définition ci-dessus exclut les langages naturels (car ils ne sont pas construits), des langages tels que Espéranto (car ils ne sont pas basés sur un langage naturel particulier), et les langages formels (car ils n'ont pas une intelligibilité intuitive).

Dans la littérature, les sous-langages (*sublanguages* en anglais) sont des termes étroitement liés aux langages contrôlés et prête souvent à des confusions. Les sous-langages sont des langages qui découlent naturellement quand une communauté de locuteurs (experts) partage des connaissances spécialisées sur un domaine sémantique restreint [et] les experts communiquent sur le domaine restreint dans une situation récurrente, ou un ensemble de situations très similaires (Kittredge, 2003). Tout comme le langage naturel contrôlé, un sous-langage est basé sur exactement un langage naturel et est plus restreint. La différence essentielle entre les deux termes est que le sous-langage émerge naturellement, alors que le langage contrôlé est explicitement et consciemment défini.

4.3.3 Les types de langages contrôlés

En général, les langages contrôlés peuvent grosso modo être divisés en fonction des problèmes qu'ils sont censés résoudre (Schwitter, 2002). Beaucoup de langages différents de ce type existent de nos jours, et sont à différents stades de maturité. (Pool, 2006) compte environ 41 projets qui définissent des sous-ensembles contrôlées de l'anglais, l'espéranto, le français, l'allemand, le grec, le japonais, le mandarin, l'espagnol et le suédois. Dans la littérature, les langages contrôlés sont en général regroupés en deux grandes catégories : Les langages contrôlés orientés-humains (*human-oriented CNL*) et les LC orientés-machines (*computer-oriented CNL*) (Huijsen, 1998). Les LCs orientés-humains visent à améliorer la communication entre les hommes (Wyner et al. 2009), c'est-à-dire améliorer la compréhensibilité et la traductibilité des textes écrits pour des lecteurs humains. En revanche, les LCs orientés-machines visent à améliorer la communication homme-machine (Wyner et al., 2009). De cette manière, ils favorisent un traitement automatique, et permettent facilement la traduction du texte contrôlé vers une logique formelle (comme ACE qui peut être traduit automatiquement et sans équivoque dans la logique de prédicat (Kuhn, 2009)), ou pour améliorer les performances des applications informatiques comme par exemple les moteurs de traduction automatique.

Une autre différence importante entre les LCs orientés-humains et les LCs orientés-machines est que les LCs orientés-humains ont des règles générales librement définies, telles que :

"Utiliser des phrases courtes", "Éviter la voix passive", "Éviter les pronoms", etc. (Temnikova, 2012). En revanche, les LCs orientés-machines, également appelés langages contrôlés basés sur la logique, sont bien définis et permettent ainsi leur traduction vers des langages formels existants. Ils sont facilement transférables vers d'autres langages de représentation de connaissances, et permettent des contrôles de cohérence automatiques (Temnikova, 2012).

Toutefois, un langage contrôlé conçu pour améliorer la communication homme-machine peut également être utilisé pour la communication entre les humains. Aussi bien, le contrôle d'un langage pour permettre une meilleure communication entre les humains peut contribuer potentiellement à améliorer son traitement automatique. Pour ces raisons, le terme « langage contrôlé » peut être utilisé dans un sens très large. Cependant, bien que dans la suite, nous présentons les deux types, le langage contrôlé RuleCNL proposé dans cette thèse est de la catégorie des LCs orientés-machines, c'est-à-dire ceux qui sont non ambigus et peuvent être mappés vers des modèles formels.

Dans la section suivante, nous présentons en détails les LCs orientés-humains et les LCs orientés-machines avec quelques exemples.

4.3.3.1 Les langages contrôlés orientés-humains (*human-oriented CNL*)

Les LCs orientés-humains ont pour objectifs d'améliorer la qualité de la communication entre les hommes à des fins spécifiques, surtout chez des locuteurs non natifs. Ils servent aussi à améliorer la lisibilité et la compréhension des documentations techniques. Exemple : *AeroSpace and Defence Simplified Technical English* (ASD STE). Ces langages contrôlés ne sont pas destinés nécessairement à un traitement par les ordinateurs, mais sont utilisés dans un objectif de bonne lisibilité et bonne compréhension par les locuteurs humains. Ils n'ont pas en général de sémantique formelle et sont généralement définis par des directives informelles. Cependant, il faut noter que certains d'entre eux initialement créés dans un but d'améliorer la lisibilité et la compréhension des documentations techniques ont évolué au fil du temps pour permettre aussi la traduction automatique comme ASD STE et *Caterpillar Technical English* (CTE). Quelques LCs de ce type sont décrits ci-dessous.

a. Basic English

Le *Basic English* (Ogden, 1930) a été créé par Charles Kay Ogden en 1930 et a une grammaire restreinte ainsi qu‟un vocabulaire de 850 mots de base les plus fréquents et familiers en anglais. Il est considéré comme un exemple de langage international auxiliaire (IAL), qui est utilisé pour la communication entre les personnes de différentes nations. Plus précisément, il a été conçu comme une base commune pour la communication en politique, en économie et dans les sciences. La version de l‟anglais contrôlé a reçu une reconnaissance plus large. Dans ce rôle, il est devenu très célèbre dans le contexte de la Seconde Guerre mondiale, comme il a été promu comme un outil pour la paix mondiale (Ogden, 1930). Les restrictions sont sans doute les plus radicales dans le cas des verbes. Seulement 18 verbes sont utilisés : *put, take, give, get, come, go, make, keep, let, do, be, seem, have, may, will, say, see,* et *send.* Ces verbes peuvent être combinés avec les prépositions pour former des relations plus spécifiques comme *put in* pour exprimer *insert.* D'autres verbes peuvent être exprimés à l'aide des noms, tels que : *give a move* au lieu d‟utiliser *move* comme un verbe. L'utilisation des mots donnés et leurs variantes est décrite par les règles de grammaire informelles, par exemple : "*Collective nouns may be formed from adjectives when used with the*."

Ci-dessous nous donnons deux exemples de phrases en *Basic English* :

The camera man who made an attempt to take a moving picture of the society women, before they got their hats off, did not get off the ship till he was questioned by the police.

It was his view that in another hundred years Britain will be a second-rate power.

De nombreuses variantes existent et utilisent des grands jeux de mots. La version anglaise simple de Wikipedia[22], par exemple, préconise d‟utiliser le *Basic English*, mais en fait, utilise un langage beaucoup moins restreint. Le *Basic English* est encore utilisé aujourd'hui et promu par le *Basic- English Institute*[23]. Beaucoup de textes ont été écrits dans ce langage, y compris des manuels, des romans, et de grandes parties de la Bible. Les simplifications drastiques sur le plan lexical, avec les restrictions grammaticales constituent un gain significatif en précision par rapport à l'anglais courant. Les restrictions informelles sur la grammaire, cependant, ne

[22] http://simple.wikipedia.org

[23] http://www.basic-english.org

sont pas assez fortes pour réduire de manière significative la complexité du langage. Le *Basic English* a influencé le *Caterpillar Fundamental English* (CFE), qui lui-même est devenu un langage très influent.

b. Caterpillar Fundamental English

Caterpillar Fundamental English (CFE) (Verbeke, 1973) était un langage contrôlé influent développé chez Caterpillar. Il a été officiellement introduit en 1971, basé sur le *Basic English* (Smart, 2003), et a été connu comme le premier LC axé sur l'industrie (Wojcik et Hoard, 1997). La nécessité de ce langage contrôlé a émergé en raison de la sophistication croissante des produits de Caterpillar et la nécessité de communiquer avec les locuteurs non anglophones des services du personnel dans différents pays (Verbeke, 1973). Le vocabulaire du langage est limité à environ 800 à 1000 mots (Crabbe, 2009), avec un seul sens défini pour chacun d'eux. Les dix règles suivantes résument les restrictions grammaticales (Crabbe, 2009) :

1. Make positive statements.
2. Avoid long and complicated sentences.
3. Avoid too many subjects in one sentence.
4. Avoid too many successive adjectives and nouns.
5. Use uniform sentence structures.
6. Avoid complicated past and future tenses.
7. Avoid conditional tenses.
8. Avoid abbreviations, contractions, and colloquialisms.
9. Use punctuation correctly.
10. Use consistent nomenclature.

Ci-dessous nous donnons deux exemples de phrases CFE:

The maximum endplay is .005 inch.
Lift heavy objects with a lifting beam only.

CFE a été interrompu par Caterpillar en 1982, parce qu'entre autres, les règles de base de la CFE n'étaient pas exécutées dans les documents en anglais produits (Kamprath et al. 1998). En conséquence, *Caterpillar Technical English* (CTE) a été élaboré à la suite d'une approche différente en 1991. Outre l'amélioration de la cohérence et la réduction de l'ambiguïté de la documentation technique, l'objectif de la CTE était d'améliorer la qualité de la traduction et de

réduire les coûts de traduction à l'aide de la traduction automatique. Contrairement à la CFE, les textes CTE sont censés être traduits avant d'être donnés au personnel dans les pays non-anglophones. Comme une autre différence, CTE vient avec un outil de création qui impose la conformité avec les restrictions. Le lexique de CTE se compose d'environ 70 000 termes. On peut donc conclure ici que CTE est de ce fait un LC orienté-machine.

c. ASD Simplified Technical English (ASD-STE)

ASD-STE, souvent abrégé STE *(Simplified Technical English)* ou tout simplement *Simplified English*, est un LC pour l'industrie aérospatiale. Il a ses origines en 1979, mais c'est seulement en 1986 qu'il a été présenté officiellement pour la première fois, sous le nom *AECMA Simplified English*. Il a reçu son nom actuel en 2004 quand AECMA a fusionné avec deux autres associations pour former ASD (*AeroSpace and Defence*). Le but principal de ce langage contrôlé est de rendre les textes plus faciles à comprendre, surtout pour les non-anglophones. Alors qu'AECMA *Simplified English* ne prenait pas en compte la traduction dans d'autres langages, l'un des objectifs initiaux de ASD-STE était de considérer cette traduction. Aujourd'hui, le langage est maintenu par le groupe *Simplified Technical English Maintenance Group*. ASD-STE est basé sur l'anglais avec des restrictions exprimées dans environ 60 règles générales. Ces règles limitent le langage au niveau lexical (par exemple, "*Use approved words from the Dictionary only as the part of speech given*"), au niveau syntaxique (par exemple, "*Do not make noun clusters of more than three nouns*"), ainsi qu'au niveau sémantique (par exemple, "*Keep to the approved meaning of a word in the Dictionary. Do not use the word with any other meaning*."). Il a un vocabulaire fixe constitué de termes communs au domaine de l'aéronautique. En outre, des noms et des verbes techniques définis par l'utilisateur peuvent être introduits. Ceci est un extrait d'un texte exemplaire dans ASD-STE :

These safety precautions are the minimum necessary for work in a fuel tank. But the local regulations can make other safety precautions necessary.

4.3.3.2 Les langages contrôlés orientés-machines (*machine-oriented CNL*)

De l'autre coté, les langages contrôlés orientés-machines ont pour objectif principal la traductibilité des documentations techniques. Ils permettent le mappage facile de textes vers des langages formels ou des représentations formelles de connaissances et à cet effet, sont

souvent reliés à des moteurs de raisonnement (Angelov et Ranta, 2009). Un langage formel est défini par (Jurafsky et Martin, 2008) comme : « *a set of strings, each string composed of symbols from a finite symbol-set called an alphabet* » et qui peut être décrit à l'aide des grammaires formelles, comme par exemple les automates finis (Mateescu et Salomaa, 1997). Ils sont complètement non ambigus et ont une représentation directe en logique formelle. Contrairement à la catégorie précédente, ces langages sont destinés à améliorer la qualité de communication entre les humains et les machines, par exemple pour l'interrogation ou l'édition d'une base de connaissance ; pour améliorer la traduction manuelle, semi-automatique ou automatique et pour fournir une représentation naturelle et intuitive pour des notations formelles. Ces derniers peuvent être définis par des grammaires formelles.

Les exemples les plus célèbres sont PENG (White et Schwitter, 2009) et ACE (Fuchs, Kaljurand et Kuhn, 2008). Les langages contrôlés orientés-machines peuvent par la suite être subdivisés en deux catégories : Les LCs génériques (*General-purpose CNL*) et les LCs construits pour des applications spécifiques.

a. Les langages contrôlés génériques

Les LCs génériques sont ceux qui sont conçus et développés sans aucun scénario spécifique ou un domaine d'application particulier. Ils ne sont pas restreints à un domaine particulier et ont été conçus pour la plupart pour servir de langages de représentation des connaissances. Ils peuvent être utilisés par des tiers pour leurs propres domaines d'application. Nous citons dans cette section quelques exemples les plus connus de ces langages.

❖ ***Attempto Controlled English*** **(**ACE) (Fuchs, Kaljurand, et Kuhn, 2008) est considéré comme le LC le plus mature initialement créé pour être mappé automatique dans une représentation logique de premier ordre. ACE a été présenté la première fois en 1996 en tant que langage de spécifications logicielles. Plus tard, l'accent s'est déplacé vers la représentation des connaissances et le web sémantique. Il a été étendu au fil des ans de diverses manières. Les caractéristiques les plus notables de l'ACE sont des phrases complexes nominales, des pluriels, des références anaphoriques, des clauses subordonnées, des modalités, et des questions.

Ci-dessous nous donnons deux phrases en ACE :

A customer owns a card that is invalid or that is damaged.

Every continent that is not Antarctica contains at least 2 countries.

ACE a un vocabulaire prédéfini et extensible. Sa grammaire définit le sens des textes dans ACE. Par exemple, "*every*" est interprété comme "*universally quantified*", comme dans la phrase "*Every cat has a tail.”*. ACE permet d'écrire des phrases simples et complexes, les phrases complexes étant composées des plus simples. Les combinaisons de phrases simples se font par la coordination, subordination, quantification, et la négation. Sa grammaire est spécifiée en *Definite Clause Grammar* (DCG) (Pereira et Warren, 1986), qui est ensuite analysée en une représentation intermédiaire sous la forme d'une structure de représentation du discours ou DRS (*Discourse Representation Structure*) (Kamp et Reyle, 1993) qui est une variante de la logique du premier ordre.

❖ ***Processable English*** **(**PENG) (White et Schwitter, 2009) est un langage contrôlé qui est semblable à ACE mais adopte une approche plus légère dans le sens où il couvre un petit sous-ensemble de l'anglais naturel. C'est un langage sans ambiguïté qui peut être traduit automatiquement via des DRS dans la logique du premier ordre avec égalité. Tout comme ACE, sa grammaire est aussi spécifiée en utilisant la notation DCG. Il est conçu pour une approche d'analyse incrémentale et a été l'un des premiers langages contrôlés utilisés avec un éditeur de texte prédictif (*look-ahead*) pour indiquer à l'utilisateur comment une phrase partielle peut être poursuivie (Schwitter et al., 2003).

D'autres langages contrôlés de ce type incluent le *Computer Processable Language* (CPL) (Clark et al. 2010) qui est un LC développé par Boeing, *Common Logic Controlled English* (CLCE) (Sowa, 2004), *Formalized-English* (Martin, 2002), etc. Une liste complète de 100 langages contrôlés basés sur l'anglais peut être trouvée dans (Kuhn, 2014). Nous avons également en Annexe III un schéma extrait dans (Kuhn, 2014) qui présente l'évolution sur le temps de ces 100 langages contrôlés.

b. Les langages contrôlés pour le web sémantique

Récemment, plusieurs LCs ont été proposés spécifiquement pour le web sémantique et qui peuvent être traduits vers des langages formels du web sémantique comme OWL, SWRL *(Semantic Web Rule Language*), etc. Parmi les plus cités on a :

❖ ***OWL ACE*** (Kaljurand et Fuchs, 2006 ; Kaljurand, 2007) est un LC pour le langage d'ontologie OWL. La syntaxe est un sous-ensemble d'ACE tandis que la sémantique est

adaptée à l'expressivité du langage OWL et est plus spécifique qu‟ACE. Ainsi, OWL ACE est plus précis mais moins expressif qu‟ACE.

❖ ***Rabbit*** (Hart et al., 2008) a été développé et utilisé par *Ordnance Survey*. Il est conçu pour un scénario spécifique, dans lequel il est utilisé pour la communication entre les experts métier et les ingénieurs d'ontologie afin de créer des ontologies pour des domaines spécifiques.

❖ ***Sydney OWL Syntax*** (SOS) (Cregan et al., 2007) est basé sur PENG et fournit une correspondance bidirectionnelle à OWL. Ainsi, les énoncés en *Sydney OWL Syntax* peuvent être traduits en expressions OWL, et vice versa.

❖ ***Controlled Language for Ontology Editing*** (CLOnE) (Funk et al., 2007) est un LC qui peut être traduit en OWL. C‟est un langage simple défini par seulement onze motifs (pattern) de phrases qui correspondent à peu près à onze axiomes OWL. Grâce à sa conception simple, seulement un petit sous-ensemble d‟OWL est couvert.

❖ ***Lite Natural Language*** (Bernardi et al., 2007), est traduit en DL-Lite, qui est un formalisme logique optimisé pour de bonnes propriétés de calcul et qui est l'équivalent d'un sous-ensemble d‟OWL.

c. Les langages contrôlés pour la traduction automatique

Comme mentionné précédemment, certains langages contrôlés conçus pour assurer la lisibilité, l'intelligibilité et la traductibilité des textes pour des lecteurs humains peuvent aussi faciliter leur traitement pour les applications informatiques. Le cas le plus fréquent étant les moteurs de traduction automatique. Des exemples de tels LCs sont donnés ci-dessous :

❖ ***Controlled Language Optimized for Uniform Translation*** (CLOUT) (Muegge 2007) est un langage contrôlé composé de 10 règles ayant pour but d‟améliorer la traduction automatique de plusieurs textes dans différents langages naturels. Les dix règles sont les suivantes :

1. *Write sentences that are shorter than 25 words.*
    - *Be brief (well structured)*
    - *Avoid ambiguity and subordination*
    - *Evaluate the information*

2. *Write sentences that express only one idea.*
    - *Simple sentences*
    - *Avoid (gerunds, juxtaposition, etc.)*
3. *Write the same sentence if you want to express the same content.*
    - *Don't be afraid to repeat*
    - *Avoid synonyms*
4. *Write sentences that are grammatically complete.*
    - *Sentences grammatically correct*
    - *Avoid ellipsis.*
5. *Write sentences that have a simple grammatical structure.*
    - *Avoid Juxtapostion, subordination, relatives pronouns, etc.*
6. *Write sentences in the active form.*
    - *Avoid passive voice*
7. *Write sentences that repeat the noun instead of using a pronoun.*
    - *Avoid pronouns like it, their, etc.*
8. *Write sentences that use articles to identify nouns.*
    - *Use the, this, etc.*
9. *Write sentences that use words from a general dictionary.*
    - *Do not use technical words*
10. *Write sentences that use only words with correct spelling.*
    - *Avoid mistakes*

❖ **LiSe (Linguistique et Sécurité)**

Le langage contrôlé LiSe a été conçu dans le projet LiSe[24] qui visait à établir à partir de la théorie micro-systémique, une méthodologie fondée sur des analyses linguistiques approfondies afin de dégager des normes linguistiques de rédaction de messages d"alertes et de protocoles pour des applications à haute sécurité, afin que ceux-ci puissent être compris et traduits aisément, rapidement et correctement. Ainsi le langage contrôlé LiSe a été conçu à la fois pour améliorer la compréhension humaine des textes et aussi en vue de faciliter leur traduction automatique du français. LiSe a été créé par le Centre de recherche en TAL Lucien

[24] http://projet-lise.univ-fcomte.fr/guide_red.html

Tesnière[25], en collaboration avec des spécialistes de l'industrie aéronautique, des professionnels de la santé et des services d'urgence, et sur la base d'une analyse manuelle préliminaire d'un corpus de documents recueillis, qui sont principalement des protocoles dans le domaine de la santé (Renahy et al., 2012). Le but de LiSe est d'aider les spécialistes de la santé à écrire clairement les documents compréhensibles par l'importation dans leur domaine des LCs qui existaient depuis longtemps dans le domaine de la documentation technique. Le langage source à partir duquel la traduction est faite est le français, et les langages de sortie sont l'anglais, le chinois, le thaï et l'arabe. Un outil d'aide à la rédaction appelé Compagnon Lise a également été développé pour faciliter l'apprentissage et l'application des règles de rédaction (Renahy et al., 2010). L'interface utilisateur du Campagon Lise est présentée à la figure 4.1. Le langage contrôlé LiSe se présente sous la forme d'un guide de rédaction et d'un dictionnaire de termes préconisés, et a été conçu afin de :

- assurer une diffusion rapide et efficace de l'information,
- assurer une compréhension parfaite du texte diffusé,
- prévenir les erreurs dues aux imprécisions et à l'ambiguïté de la langue naturelle, aux situations d'urgence ou de stress,
- permettre une traduction automatique correcte et fiable vers 3 langues : anglais, arabe et chinois,
- obtenir des textes uniformisés, conformes à des modèles et des normes établis avec des professionnels.

[25] http://tesniere.univ-fcomte.fr

Figure 4.1 : Interface utilisateur du Compagnon Lise

Pendant le projet *MESSAGE*[26], le langage contrôlé LiSe a été transféré à trois autres langues européennes : espagnol (Blanco, 2009), anglais (Temnikova et Orasan, 2009) et polonais (Cholewa, 2009 ; Gwiazdecka, 2009 ; Rudas 2009). Des recherches ont également été menées en faveur du développement de CL prototypes pour les situations d'urgence concernant le grec moderne (Papadopoulou et Portella, 2009) et le bulgare (Temnikova et Margova, 2009).

4.4 Les langages contrôlés pour les règles métier

Cette thèse consiste à proposer un langage contrôlé pour la spécification des règles métier et leurs transformations automatiques vers le standard SBVR. Dans ce qui précède, nous avons présenté une vue d'ensemble sur les langages contrôlés ainsi que les différentes applications et domaines auxquels ils sont appliqués. En réalité, il existe dans la littérature un nombre important de langages contrôlés naturels pour interfacer les langages formels de représentation des connaissances et des ontologies dans le champ du web sémantique. Toutefois, il n'en existe pas assez pour la spécification des règles métier de façon naturelle à l'intention des experts métier (couche CIM de la figure 4.2). Dans cette section, nous focalisons notre attention sur quelques travaux sur les langages contrôlés pour la spécification

[26] http://message-project.univ-fcomte.fr/index-fr.html

des règles métier de façon naturelle. La figure 4.2 présente les langages de règles à différents niveaux d'abstractions de l'architecture du *Model Driven Architecture*.

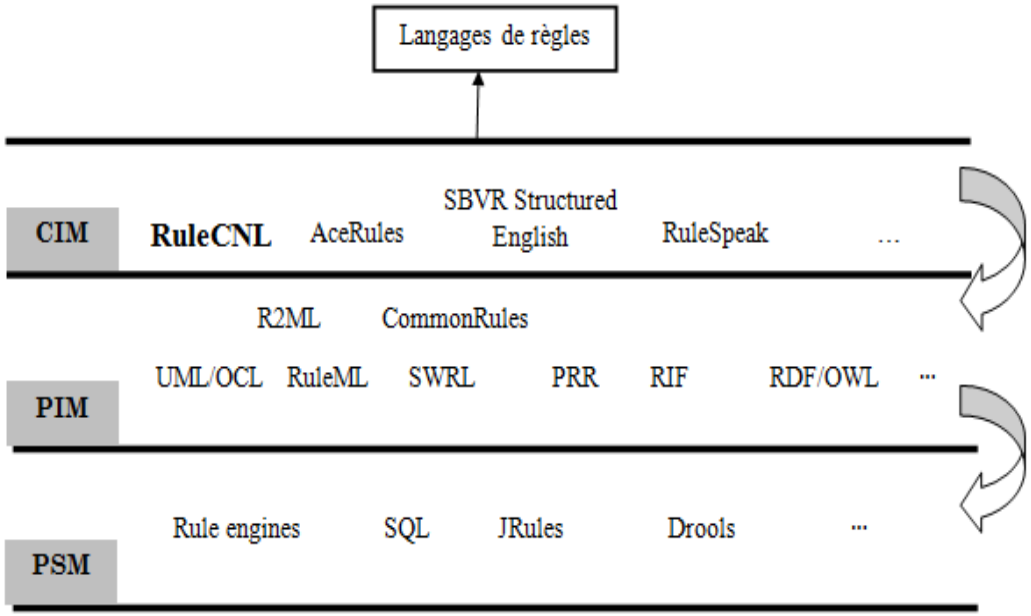


Figure 4.2 : Hiérarchie des langages de règles dans un contexte MDA

La plupart des travaux existants qui consistent à utiliser des représentations en langage naturel pour les systèmes de règles sont basés sur l'idée de verbaliser les règles qui existent déjà dans une représentation formelle, c'est-à-dire aller des couches *Platform Specific Model* et *Platform Independent Model* vers la couche *Computation Independent Model* de la figure 4.2 (Halpin, 2004 ; Jarrar, Keet et Dongilli, 2006 ; Lukichev et Wagner, 2006). Plus particulièrement dans le contexte du projet Européen REWERSE (*Reasoning on the Web with Rules and Semantics*)[27] qui a duré 4 ans (2004 à 2008) impliquant 27 laboratoires de recherche universitaires et industrielles de 17 pays européens différents, des travaux de recherches (Wagner et al., 2005 ; Nicolae et Wagner, 2006) se sont tournés vers la verbalisation des règles exprimées dans les langages visuels de règles (UML) et textuels (OCL, RDF, OWL), la verbalisation des règles exprimées en langage R2ML vers SBVR SF et SBVR SE. R2ML (Nicolae et Wagner, 2006) est un langage de règles basé sur XML développé pendant ce projet qui prétend être complet dans le sens où il intègre : le langage OCL qui est un standard

[27] http://rewerse.net/index.html

utilisé dans l'ingénierie des systèmes d'information et de génie logiciel ; le langage SWRL (*Semantic Web Rule Language*) qui est une proposition visant à étendre le OWL en ajoutant des axiomes d'implication ; et le langage RuleML qui est une proposition de *RuleML Initiative* basée sur Datalog/Prolog. Toujours dans ce même contexte, le système AceRules[28] est un système de règles utilisant le langage contrôlé ACE pour formaliser les règles et les faits (Kuhn, 2007). AceRules utilise le LC générique ACE et de ce fait ce système se base sur un interpréteur à chainage avant qui calcule les conclusions possibles pour un ensemble de règles et de faits. Il dispose d'une interface web simple (cf. figure 4.3) qui est conçue pour cacher les détails techniques en se basant sur le LC ACE. Fondamentalement, cette interface est constituée de deux zones de textes. La première nommée « Program » peut être utilisée pour écrire du texte libre qui est interprété selon les règles définies dans ACE. La seconde nommée « Answer » qui est en lecture seule reçoit le résultat inféré à partir du texte entré et le résultat est toujours dans une syntaxe ACE. Ainsi, AceRules est plus un système de raisonnement ou de déduction car il suit strictement la tradition des langages de règles du style logique de prédicat comme Prolog. En plus, il ne fournit aucune assistance à la rédaction des règles. L'utilisateur doit avoir une bonne connaissance du LC ACE pour pouvoir l'utiliser ou comprendre la logique cachée derrière. Lorsque l'utilisateur entre quelque chose qui n'est pas correct selon la syntaxe de ACE ou ne peut pas être transformé en une structure de règle interne alors un message d'erreur est affiché à l'utilisateur simplement. Ces messages d'erreur tentent d'expliquer la raison pour laquelle le programme donné n'a pas pu être exécuté, mais il pourrait être difficile du moins pour des utilisateurs inexpérimentés d'être en mesure de résoudre le problème.

[28] http://attempto.ifi.uzh.ch/acerules

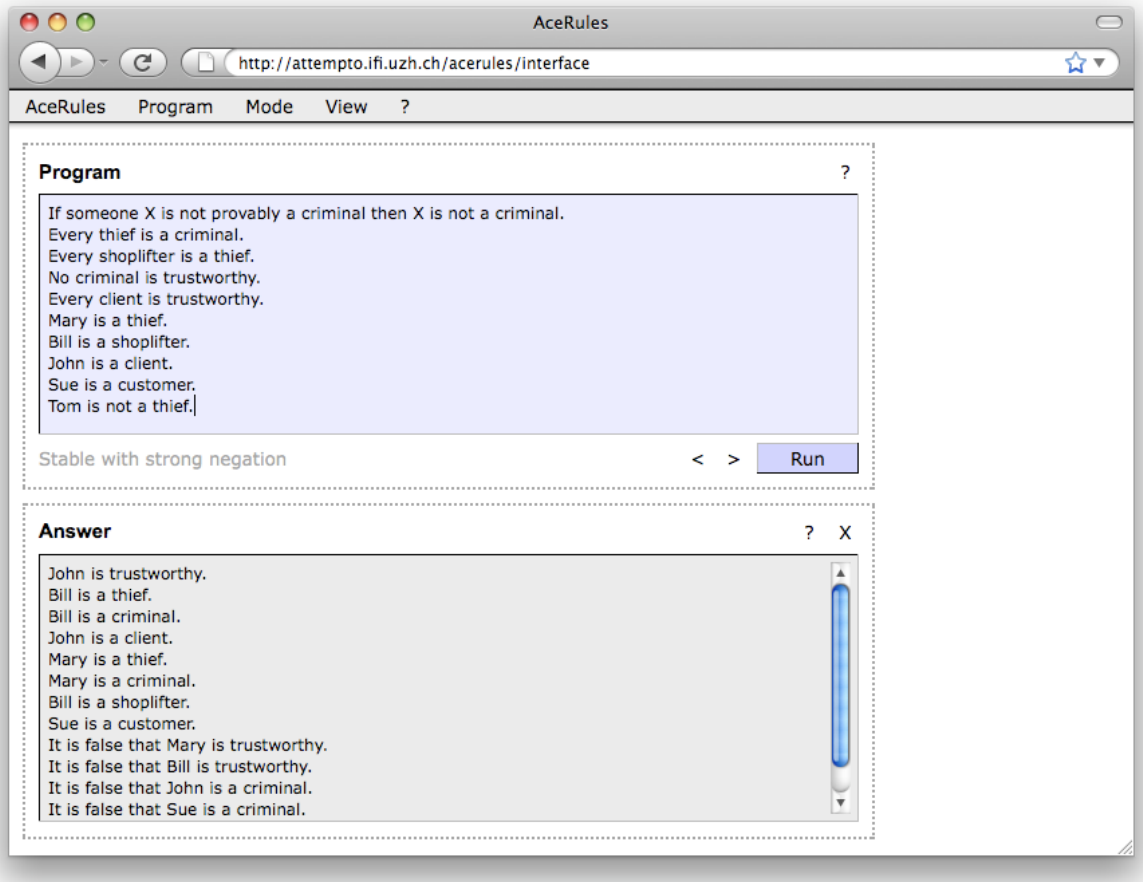


Figure 4.3. L‟interface web du système AceRules

SBVR Structured-English (OMG, 2008a) et *RuleSpeak* sont définis dans la spécification SBVR à titre non normatif et cités parfois dans la littérature comme des langages contrôlés pour exprimer des règles métier avec une sémantique basée sur SBVR. Toutefois, il faut noter que la spécification SBVR ne standardise pas un langage de règles métier particulier pour exprimer le vocabulaire et les règles métier, mais spécifie un méta-modèle (syntaxe abstraite indépendante de tout langage) pour définir le modèle (structure) sémantique du vocabulaire et des règles métier exprimés en langage naturel utilisé par les experts métier (cf. Section 3.5.1 du Chapitre 3). En d‟autres termes, il établit un vocabulaire pour parler du sens ou de la structure sémantique du vocabulaire et des règles métier. La majeure partie du méta-modèle appelée Formulation sémantique permet effectivement de décrire la structure du sens (composition logique de sens) des règles et des définitions des concepts pour permettre une interprétation cohérente des concepts et des règles.

RuleSpeak (Ross, 2003 ; OMG 2008a ; Ross, 2013) est défini comme un ensemble de directives pour exprimer des règles métier de façon concise, Ce n'est pas un langage ou une syntaxe en soi, mais plutôt un ensemble de bonnes pratiques pour les locuteurs de l'anglais[29]. Le développement du langage a commencé en 1985 et il a été présenté la première fois en 1994. Il est très similaire au *SBVR Structured-English*, qui a émergé plus tard aux alentours de 2005. Tous les deux utilisent une version restreinte de l'anglais et peuvent être mis en correspondance de façon informelle avec la formulation sémantique du méta-modèle SBVR.
Ainsi, *SBVR Structured-English* et *RuleSpeak* ne sont pas des langages en soi, mais plutôt un ensemble de bonnes pratiques pour les locuteurs humains. Ils sont définis de façon informelle par des ensembles de propositions fondées sur les expériences de bonnes pratiques dans les systèmes de règles (Kuhn, 2010). Ils n‟ont pas de syntaxe (grammaire formelle) et par conséquent ne sont supportés par aucun outillage et ne peuvent être traités de manière entièrement automatique (Ruth, 2009). La structure de la syntaxe peut être ambiguë et est obtenue par une mise en forme manuelle des styles et couleurs de polices de caractères, ce qui pourrait être utilisé pour faciliter la compréhension du lecteur humain. Ces restrictions et ces directives restent très importantes pour un lecteur humain, car *SBVR Structured-English* et *RuleSpeak* pourraient être considérés comme des LCs orientés-humains. Cependant, des efforts supplémentaires doivent être fournis afin de les formaliser (définir une grammaire formelle) pour un traitement automatique. Sur cette base, nous proposons dans cette thèse un LC orienté-machine pour la spécification des règles métier et un moteur de transformation vers les structures sémantiques SBVR.

4.5 Conclusion

Dans ce chapitre, nous avons présenté les langages contrôlés dans un ensemble général. Nous avons présenté les origines ainsi que les différents types de LCs existants. Après avoir présenté les LCs orientés-humains avec les exemples les plus cités de la littérature, nous nous sommes focalisés sur ceux orientés-machines qui sont ceux qui nous intéressent dans cette thèse. Ces derniers sont ceux qui peuvent être mappés automatiquement (et souvent de façon déterministe) vers des langages formels. Ces LCs peuvent être utilisés dans plusieurs domaines ainsi que plusieurs applications. Plus précisément dans le contexte des SI, ils peuvent être utilisés pour exprimer différents types d‟informations comme les règles métier.

[29] http://www.rulespeak.com/en

Dans la littérature, bien que des langages/systèmes formels existent en quantité avec un niveau de maturité pour soutenir l'ARM, il se pose encore le problème d'interface en langage naturel à ces systèmes formels pour permettre aux experts métier sans connaissance techniques de pouvoir spécifier ou valider leurs règles métier. Les LCs orientés-machines peuvent sans doute apporter leur contribution dans ce sens. Dans le chapitre suivant, nous allons présenter notre langage contrôlé RuleCNL qui est notre contribution dans cette direction.

Chapitre 5

RuleCNL : notre langage contrôlé pour la spécification des règles métier

5.1 Introduction

Nous avons présenté au Chapitre 4 un état de l'art sur les langages contrôlés. Nous avons vu que les langages contrôlés et plus précisément les langages contrôlés orientés-machines sont de bons médiateurs entre les langages naturels des humains et les langages formels des machines. Ils peuvent ainsi être conçus pour servir à des buts spécifiques et pour des applications différentes. Dans ce chapitre, nous présentons notre langage contrôlé appelé RuleCNL pour la spécification des règles métier dans le contexte de modélisation des systèmes d'informations. Comme nous avons mentionné au Chapitre 1, notre méthodologie est basée sur l'alignement de la définition des règles métier avec le vocabulaire métier. Ainsi, les règles métier sont sémantiquement liées au domaine métier et facilement compréhensibles par les experts du domaine. Cette méthodologie est dérivée du principe de base de l'ARM préconisé par le BRG comme suit (BRG, 2000) :

"Rules build on facts, and facts build on concepts as expressed by terms."

Afin de surmonter les limitations soulignées dans les langages contrôlés pour règles métier comme *SBVR Structured-English* et *RuleSpeak* présentés à la Section 4.4, nous avons défini une grammaire formelle et par conséquent un parseur qui peut être utilisé pour l'analyse syntaxique des règles dans RuleCNL. Le processus d'écriture d'une règle métier est entièrement sous le contrôle de la cohérence imposée par notre méthodologie. La sémantique d'exécution de RuleCNL est définie par des transformations automatiques vers des formulations sémantiques SBVR. Ces dernières permettent de décrire la structure sémantique des énoncés de règles indépendamment du langage et ont une base théorique fondée sur la logique formelle.

5.2 Approche de conception

Comme nous avons déjà souligné à la Section 1.4.2, la mise en œuvre de notre RuleCNL suit une approche de conception dirigée par les modèles. L'approche MDA présentée au Chapitre 3 permet la montée en abstraction grâce aux modèles divisés en trois niveaux d'abstractions. Ainsi, la conception des modèles/langages de règles métier peut aussi suivre cette approche comme le montre l'architecture générale de RuleCNL présentée à la figure 1.3. La mise en œuvre de RuleCNL est basée sur ses deux principes clés à savoir : la méta-modélisation et la transformation des modèles.

5.3 Méta-modélisation de RuleCNL

RuleCNL est le langage contrôlé que nous proposons dans cette thèse pour la spécification des règles métier dans le contexte de modélisation des SI. Que ce soit en linguistique (langage naturel) ou en informatique (langage de programmation ou de modélisation), un langage est caractérisé de façon générale par sa syntaxe, son domaine sémantique et la correspondance entre la syntaxe et le domaine sémantique (Harrel et Rumpe, 2004 ; Combemale, 2008 ; Kahlaoui, 2011). La syntaxe est par la suite composée de la syntaxe abstraite et de la syntaxe concrète. À la Section 3.4.1, nous avons vu selon le principe de la méta-modélisation que la syntaxe abstraite peut être spécifiée par le biais d'un méta-modèle qui définit la structure commune à tous les modèles. Elle constitue également un pilier pour la transformation des modèles. Un méta-modèle est défini à l'aide du langage de méta-modélisation MOF/UML qui est un sous-ensemble du diagramme de classe UML (cf. figure 3.4) et recommandé par l'OMG comme le langage standard de tous les méta-modèles. Le méta-modèle de RuleCNL qui décrit les concepts du langage RuleCNL (son vocabulaire), leurs propriétés ainsi que les relations entre eux est défini selon ce principe de méta-modélisation qui s'adapte bien aux techniques de spécifications formelles (OMG, 2006a ; 2007). La syntaxe concrète ou la syntaxe de surface offre un moyen convivial aux utilisateurs du langage à travers des notations appropriées pour spécifier les modèles du langage. Cette dernière peut prendre plusieurs formes : textuelle, graphique, tabulaire, visuelle, etc. Dans le contexte de cette thèse, le langage RuleCNL que nous proposons est un sous-ensemble du langage naturel des experts métier. Ainsi, sa syntaxe concrète est textuelle comme celle du langage naturel. De ce fait, nous définissons une grammaire formelle pour la définition des règles de sa syntaxe. La figure 5.1 ci-dessous présente l'architecture générale de la méta-modélisation de RuleCNL.

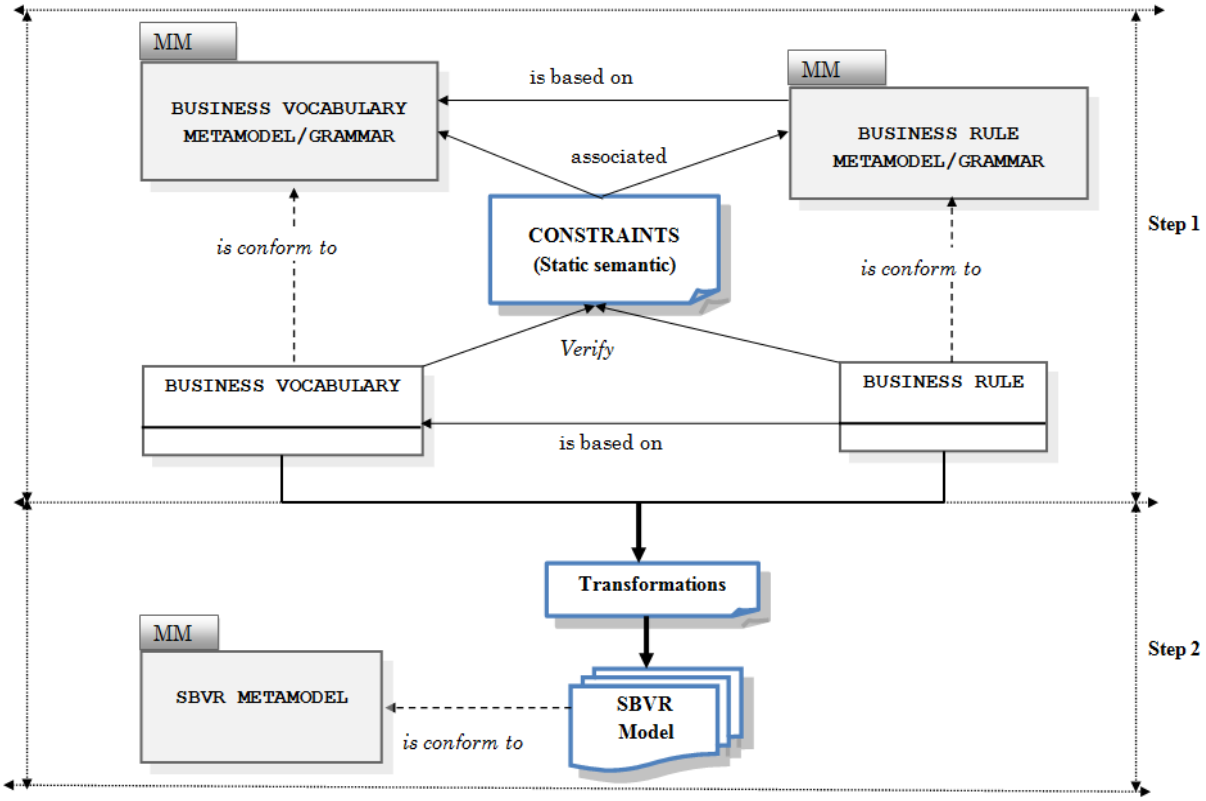


Figure 5.1 : Architecture de méta-modélisation de RuleCNL

Le fait d'adopter une approche de méta-modélisation pour la mise en œuvre de notre langage contrôlé RuleCNL offre plusieurs avantages. Cette approche permet de définir une syntaxe abstraite de RuleCNL qui pourrait être utilisée pour la définition de plusieurs langages concrets pouvant servir à des fins différentes. Par exemple, une application pourrait être d'utiliser une syntaxe concrète basée sur XML pour le web. La plupart des langages contrôlés présentés au Chapitre 4 sont conçus suivant une approche orientée-syntaxe (textuelle) tandis que nous avons une approche orientée-modèle (concept). Cette approche favorise la maintenance, la modularité et la réutilisation dans la définition des syntaxes concrètes pour des communautés d'utilisateurs spécifiques et à des fins particulières. D'après la figure 5.1 ci-dessus, nous voyons que le méta-modèle de RuleCNL est divisé en deux parties.

Nous avons tout d'abord le méta-modèle du vocabulaire métier (*Business Vocabulary Metamodel*) c'est-à-dire le vocabulaire de RuleCNL, soit le lexique des concepts du domaine métier, les propriétés de ces concepts ainsi que les relations entre ces concepts. Ainsi, la définition du vocabulaire d'un domaine métier donné sera conforme à ce méta-modèle. Une

grammaire est également associée à ce méta-modèle pour définir la syntaxe textuelle qui désigne la notation utilisée pour le vocabulaire. Plus précisément, RuleCNL exige que les termes métier soient déclarés et définis conformément au méta-modèle du vocabulaire métier. En effet, les termes dans un langage contrôlé peuvent avoir un sens très précis et pas nécessairement tel que défini dans le dictionnaire du langage naturel. Par exemple, dans un contexte général, les termes « Client » et « Customer » peuvent être considérés comme des synonymes en anglais. Cependant, dans une entreprise particulière, le terme «Client » pourrait signifier une personne qui n'a pas un compte avec l'entreprise, tandis que le terme «Customer» pourrait signifier une personne qui possède un compte avec la dite entreprise. Toutefois, l'outil support de RuleCNL assiste l'expert métier dans la déclaration et la définition des termes métier.

Ensuite, nous avons le méta-modèle des règles métier (*Business Rule Metamodel*) c'est-à-dire le méta-modèle proprement dit de RuleCNL qui structure les concepts des règles métier. Ainsi, la définition des règles métier sera conforme à ce méta-modèle. De par notre méthodologie, le méta-modèle des règles métier fait référence au méta-modèle du vocabulaire métier. Une grammaire est également associée à ce méta-modèle pour définir la syntaxe textuelle pour la notation des règles métier. La définition d'une règle métier suppose l'existence d'un vocabulaire métier ou de domaine, ce qui signifie que chaque terme métier associé à un concept de RuleCNL est en quelque sorte déclaré ou défini dans le vocabulaire de RuleCNL.

À coté de ces deux méta-modèles, nous avons défini un ensemble de contraintes portant sur la sémantique (statique) de RuleCNL. Ces contraintes doivent être vérifiées lors de la spécification et validation du vocabulaire et des règles métier. Elles sont spécifiées de façon textuelle et sont celles qui ne peuvent pas être directement spécifiées dans les diagrammes de ces méta-modèles. Par exemple, une contrainte basique impose que la définition d'un terme métier désignant un nom propre doit commencer par une lettre majuscule.

Enfin, grâce au principe de transformations des modèles, nous définissons la sémantique d'exécution de notre RuleCNL qui consiste à transformer les modèles du vocabulaire et des règles métier de RuleCNL vers des modèles sémantiques SBVR. Ces modèles sémantiques SBVR sont conformes au méta-modèle SBVR. Le méta-modèle SBVR a été présenté à la

Section 3.5. Nous présentons dans les sections suivantes ces méta-modèles qui définissent la syntaxe de notre langage contrôlé RuleCNL.

5.3.1 Méta-modèle du vocabulaire métier dans RuleCNL

Quand on parle de la syntaxe d'un langage, il est d'usage de parler du vocabulaire (le lexique) du langage ainsi que de la structure du langage (la grammaire), c'est-à-dire, la manière dont les phrases sont composées en utilisant des éléments du vocabulaire. Dans le cas des langages de règles métier, le vocabulaire est constitué du vocabulaire de l'entreprise encore appelé le vocabulaire métier, qui est spécifique au domaine d'activité. Ainsi, le vocabulaire métier dans RuleCNL décrit le modèle conceptuel d'un domaine ou l'ontologie de l'entreprise. Il définit un ensemble cohérent de concepts reliés entre eux (termes du domaine et leurs relations) qu'une entreprise donnée utilise à l'oral et à l'écrit dans le cadre de ses activités. Il est défini de façon structurée par un expert métier et représente l'univers de discours du métier, c'est-à-dire les connaissances qu'une entreprise possède sur elle-même.

Ce modèle conceptuel est un modèle d'objet qui reflète la vue du métier sur les données/objets de l'entreprise. Il représente l'environnement métier comme un ensemble d'objets, avec des caractéristiques, à la fois structurelles (attributs) et comportementales (fonctions), et aussi avec des relations entre eux. Ce modèle est orienté métier dans le sens qu'il reflète le point de vue des experts métier et non celui des experts système. Il peut être assimilé à un diagramme de classe UML, où les termes correspondent aux classes et leurs attributs, et les faits (relations) correspondent aux associations. Bien que ce soit une bonne approximation, il est important de savoir qu'il ya des différences conceptuelles entre les deux. La spécification des règles métier fait référence aux termes métier et aux faits qui sont généralement implémentés par ce modèle objet métier qui incarne la vue du métier sur des données de l'entreprise. Ce modèle doit satisfaire quelques propriétés citées ci-dessous :

a. Spécificité. La terminologie des termes du modèle devrait suivre de près celle utilisée par le domaine métier concerné. Par exemple, si nous avons affaire à des règles métier du domaine de la banque, nous parlerons de *Compte Bancaire*, *Client*, *Carte Bancaire*, *Compte Courant*, etc. Si nous avons affaire à un domaine universitaire, nous aurons un vocabulaire avec *Etudiant*, *Note*, *Cours Magistral*, *Contrôle Continu*, *Examen*, etc.

b. Abstraction. Le modèle doit représenter l'information à un niveau d'abstraction approprié pour la spécification des règles métier. Par exemple, si la règle métier parle d'ˇun «Montant de Prêt Bancaire », alors le modèle doit avoir un attribut Montant pour la classe Prêt Bancaire, même si cette information doit être agrégée à partir des autres données.

c. Pertinence. Idéalement, le modèle ne doit représenter que l'information qui est pertinente à une fonction métier en particulier (par exemple, un processus métier).

d. Complet. Le modèle doit avoir toutes les informations nécessaires pour la fonction métier et rien de plus.

La figure 5.2 présente le méta-modèle du vocabulaire métier de RuleCNL qui doit permettre de définir les modèles conceptuels du vocabulaire métier.

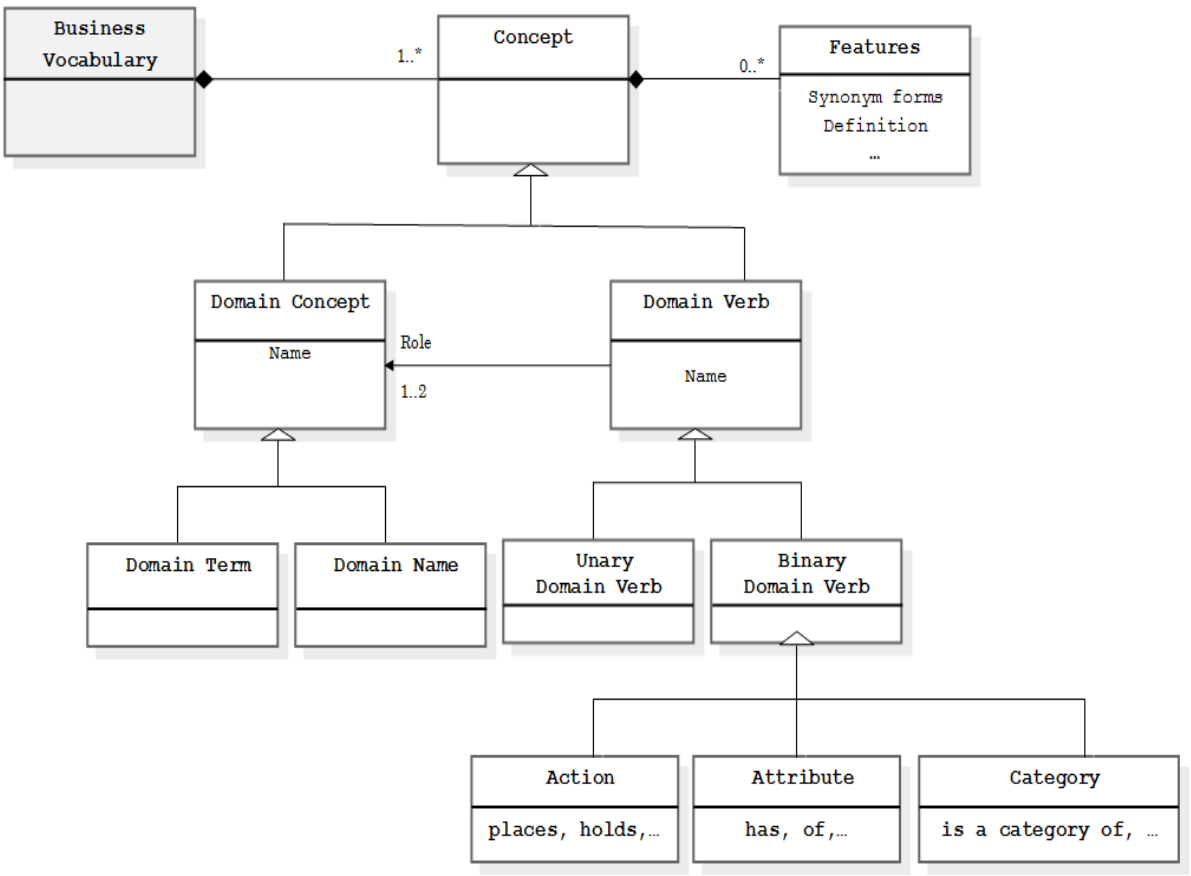


Figure 5.2 : Méta-modèle du vocabulaire métier

Comme le montre la figure 5.2, ce méta-modèle décrit un vocabulaire métier (du domaine) qui est constitué d'au moins un ou plusieurs concepts spécialisés dans le domaine métier considéré. Ces concepts peuvent être soit des termes métier soit des verbes qui représentent des relations entre les termes métier. Ces concepts sont désignés par des mots, des expressions ou des syntagmes spécifiques. Ils peuvent avoir des propriétés supplémentaires comme des synonymes, des définitions, etc.

Remarque. Pour permettre une meilleure lisibilité, dans les exemples donnés dans la suite de ce chapitre, les *Domain Concepts* sont ainsi soulignés et les *Domain Verbs* sont en italique. Les concepts de règles (*Rule Concepts*) que nous présentons plus loin dans la Section 5.3.2.2 sont ainsi mis en forme en gras. Les exemples seront indifféremment en anglais et en français.

5.3.1.1 Domain Concept

Un *Domain Concept* peut soit designer une classe d'objets métier (*Domain Term*), soit un objet spécifique (*Domain Name*).

a. *Domain Term*

Un *Domain Term* désigne une entité métier significative qui peut être représentée par un nom commun ou un syntagme nominal. Un *Domain Term* est toujours représenté sous une forme singulière et sans articles ou déterminants. Ce sont des noms sémantiquement pleins. La syntaxe de déclaration d'un *Domain Term* suit la forme :

TermDeclaration = "Term:", DesignationTerm

Exemple :

Term: client

Term: compte bancaire

Term: âge de départ à la retraite

La déclaration d'un *Domain Term* peut comporter des entrées supplémentaires correspondant à ses propriétés telles que la forme synonyme, la définition (informelle) du terme. Dans le cas où il y a un synonyme, alors la désignation du terme devient la désignation principale.

Exemple :

Term: client,

Synonym: customer

Definition: “A person who has an account”

b. *Domain Name*

Un *Domain Name* désigne une entité métier significative qui représente seulement une chose. Ce sont généralement des noms propres. Leur désignation commence toujours par des lettres majuscules. La syntaxe de déclaration d‟un *Domain Name* suit la forme :

NameDeclaration = "Name:", DesignationName

Exemple :

Name: France

Term: Euro

Term: USA

5.3.1.2 Domain Verbs

Un *Domain Verb* ou un *verbe métier* ou encore *un fait* désigne une relation, une situation, une caractéristique ou une action impliquant un ou deux termes métier. Dans un souci de garder le vocabulaire de RuleCNL simple et facilement manipulable par des experts métier, nous ne considérons que les verbes métier unaires et binaires.

a. *Domain Verb* binaire

Le *Domain Verb* binaire définit une relation sémantique et possède deux rôles remplis par les termes métier. La syntaxe de déclaration d‟un *DomainVerb* binaire suit la forme :

VerbDeclaration = "Fact Type:", Role1, DesignationVerb, Role2

Exemple :

Fact type: client *possède* compte bancaire

Fact type: compte bancaire *possède* montant

Le *Domain Verb* binaire en soi n'est qu'une partie de la syntaxe de cette déclaration et n'a pas de sens de façon isolée, mais seulement dans la relation.

Par exemple : si nous prenons le verbe *to run*, il a des significations différentes dans les relations suivantes :

Fact Type: manager *runs* company

Fact Type: horse *runs* race

Fact Type: computer *runs* program

Un *Domain Verb* binaire peut être écrit à la fois à la voix active et passive. Dans ce cas, la voix active sera la désignation principale et la voix passive sera une propriété.

Fact Type: client *possède* compte bancaire

Synonym Form: compte bancaire *est possédé par* client

Le *Domain Verb* binaire peut être un verbe au sens linguistique du terme (partie du discours). Dans ce cas, il est toujours conjugué à la troisième personne du singulier de l‟indicatif (exemple : *possède*, *achète*, etc.). Dans le cas où la troisième personne du singulier de l‟indicatif est différente de la troisième personne du singulier du subjonctif, alors cette dernière est spécifiée comme une propriété (exemple : *prend* et *prenne*, etc.).

Le *Domain Verb* peut également être la combinaison d'un verbe avec des mots fonctionnels (prépositions, etc.). Exemple : *cohérent avec*, etc.

Nous établissons par la suite une taxonomie du *Domain Verb* binaire en trois sous catégories :

➢ Les « *Action Domain Verb* » qui désignent les *Domain Verbs* qui sont des relations sémantiques indiquant une action entre le *Role1* et le *Role2* dans la syntaxe de déclaration.

Exemple : client *commande* article

➢ Les « *Attribute Domain Verb* » qui désignent les *Domain Verbs* qui sont des relations sémantiques indiquant que le *Role2* est un attribut de *Role1* dans la syntaxe de déclaration.

Exemple : compte bancaire *a* solde

Le *Domain Verb* est représenté dans ce cas généralement par le verbe *avoir* ou *posséder*. Ainsi, l'usage du verbe *avoir* doit être limité à des attributs (propriétés directes) ou des caractéristiques d'un *Domain Concept*. Le verbe *avoir* est imprécis lorsqu'il est utilisé pour exprimer une relation entre des *Domain Concepts* indépendants car il obscurcit le vrai sens de la relation.

➢ Les « *Category Domain Verb* » qui désignent les *Domain Verbs* qui sont des relations sémantiques indiquant que le *Role1* est un type ou une catégorie de *Role2* dans la syntaxe de déclaration.

Exemple : client fidèle *est un* client ; gold customer *is a* customer ; etc.

Le *Domain Verb* est représenté dans ce cas généralement par le verbe *est un (e)* ou *est une catégorie de*, etc.

b. *Domain Verb* unaire

Le *Domain Verb* unaire définit une caractéristique ou un état d'un terme métier qui remplit la fonction d'un rôle. Son évaluation conduit à une valeur booléenne. La syntaxe de déclaration d'un *Domain Verb* unaire suit la forme :

VerbDeclaration = "Fact Type:", role, DesignationVerb

Exemple :

Fact Type: order *shipped*

Fact Type: customer *smokes*

Il n'y a pas de mots supplémentaires ou de mots fonctionnels dans la déclaration d'un *Domain Verb* ce qui permet une grande souplesse car les contraintes ou les restrictions sur les termes métier ou les verbes métier sont ajoutées lors de la définition des règles métier. Le vocabulaire de RuleCNL inclut certaines relations prédéfinies comme les verbes de comparaison (égalité/inégalité) qui ne sont pas définies par les experts métier. Ces experts

métier définissent ou importent leur vocabulaire métier avec l'aide de l'éditeur du vocabulaire de l'outil support de RuleCNL.

La figure 5.3 ci-dessous récapitule un exemple de vocabulaire que nous venons de décrire.

```
Term: customer
Term: order
Term: account
Term: age
Term: outstanding_balance

Fact Type: customer places order
Fact Type: account has outstanding_balance
Fact Type: customer holds account
Fact Type: customer has age

Fact Type: order shipped
Fact Type: customer adult
```

Figure 5.3 : Exemple de vocabulaire métier dans l'éditeur du vocabulaire

5.3.2 Méta-modèle des règles métier dans RuleCNL

Conceptuellement, la syntaxe abstraite (méta-modèle) définit la structure d'une règle métier dans notre langage RuleCNL. D'une manière générale, une règle métier est composée des faits et des concepts de règles (*Rule Concepts*) comme indiqué sur la figure 5.4. Un fait est construit à partir des termes métier et exprime la relation entre eux. La déclaration d'une règle métier conforme à notre méta-modèle est définie de façon structurée en combinant des *Rule Concepts* et des *Domain Verbs* ou des faits. Ces derniers sont préalablement définis en utilisant des termes métier (*Domain Concept).* Les faits et les termes métier sont définis dans le vocabulaire métier comme nous avons présenté plus haut. Comme l'impose notre méthodologie, le méta-modèle de RuleCNL suppose l'existence d'un vocabulaire métier ou de domaine, aussi, la spécification d'une règle suppose que chaque terme du langage est en quelque sorte déclaré dans le vocabulaire de RuleCNL. Les seuls termes que l'outil support de RuleCNL comprend sont les *Rule Concepts* définis dans le méta-modèle. Ainsi, afin d'éviter toute ambiguïté, le vocabulaire doit être défini au préalable pour assurer la cohérence dans la spécification des règles métier.

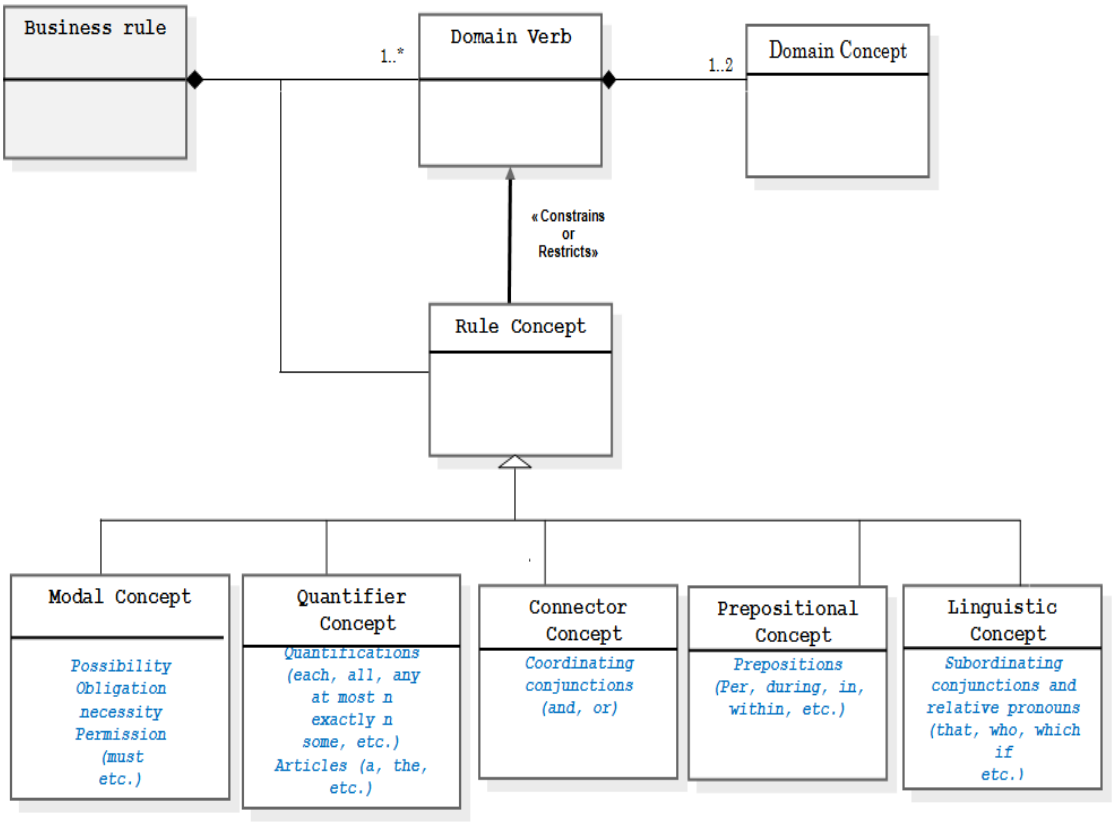


Figure 5.4 : Modèle conceptuel d'une règle métier dans RuleCNL

Tous les faits pertinents doivent être identifiés avec leurs formes synonymes si nécessaire et explicitement inclus dans la déclaration des règles métier. En effet, les ambiguïtés dans une règle résultent inévitablement lorsque des faits pertinents sont omis dans sa déclaration. Seuls les faits prédéfinis qui expriment les opérateurs de comparaison (*est supérieure à*, *est inférieure à*, etc.) peuvent être omis et n'ont pas besoin d'être inclus dans le vocabulaire métier.

Par exemple, considérons la règle suivante :

The invoice must be paid within 30 days of being issued.

Cette règle exprimée ainsi est très ambiguë. Elle implique l'existence de plusieurs faits non exprimés. Il s'agit ici d'une contrainte dynamique imposant des restrictions sur les transitions possibles (dates) entre les états d'une entité métier (*Domain Term*). La contrainte peut simplement comparer un état à un autre, elle doit alors être formellement exprimée dans RuleCNL de la façon suivante :

- Identification des *Domain Concepts* : invoice, date
- Identification des *Domain Verbs* (faits) : invoice *is issued on* date ; invoice *is paid on* date
- Spécification formelle de la règle :

  It is obligatory that each invoice **that** *is issued on* date „$Date_1$" *is paid on* date „$Date_2$„ **where** $Date_2$ *is equal or less than* $Date_1$ **plus** 30 days.

On peut voir ici qu'on a un fait de comparaison prédéfini à savoir :

$quantity_1$ *is equal or less than* $quantity_2$

5.3.2.1 *Domain Verb* et *Domain Concept*

Sur la figure 5.4, les éléments *Domain Verb* et *Domain Concept* sont décrits respectivement à la Section 5.3.1.2 et Section 5.3.1.1.

5.3.2.2 Rule Concept

Le *Rule Concept* désigne des concepts de règles métier qui sont combinés avec des faits du vocabulaire afin de former une déclaration de règle métier grammaticalement correcte. Il est divisé en cinq sous catégories présentées ci-dessous :

➢ Le « *Modal Concept* » qui indique la modalité de la règle c'est-à-dire si la règle exprime une nécessité, une obligation, une interdiction ou une possibilité.

➢ Le « *Quantifier Concept* » qui désigne les déterminants (les quantificateurs et les articles) pouvant être combinés avec des *Domain Concepts* pour former des syntagmes nominaux dans la déclaration d'une règle métier.

➢ Le « *Connector Concept* » qui désigne des conjonctions de coordination (*et*, *ou*) permettant de relier des clauses simples pour former des clauses composées dans le cas des règles métier complexes et composées.

➢ Le « *Prepositional Concept* » qui désigne des prépositions permettant de former des clauses prépositionnelles.

➢ Le « *Linguistic Concept* » qui désigne des constructions linguistiques : des conjonctions de subordination (*si*) permettant d'exprimer des conditions ; des pronoms relatifs (*qui*) qui

introduisent des clauses relatives utilisées pour qualifier ou contraindre une clause principale dans une règle métier.

5.3.2.3 Représentation d'une règle métier dans RuleCNL

À la Section 2.2.4, nous avons présenté la classification générale des règles métier qui constitue une phase très importante dans l'ARM. Plus précisément, nous avons établi à la Section 2.2.4.3, notre classification de règles métier qui étend la classification de l'OMG avec des règles structurelles et opérationnelles (voir figure 2.4). Cette classification en amont constitue une bonne conception pendant le cycle de vie des règles métier et facilite leur implémentation. Toutefois, notre attention est plus focalisée sur les structures de représentation d'une règle métier. Ainsi la figure 5.5 présente la structure générale c'est-à-dire le méta modèle pour la représentation d'une règle métier dans RuleCNL.

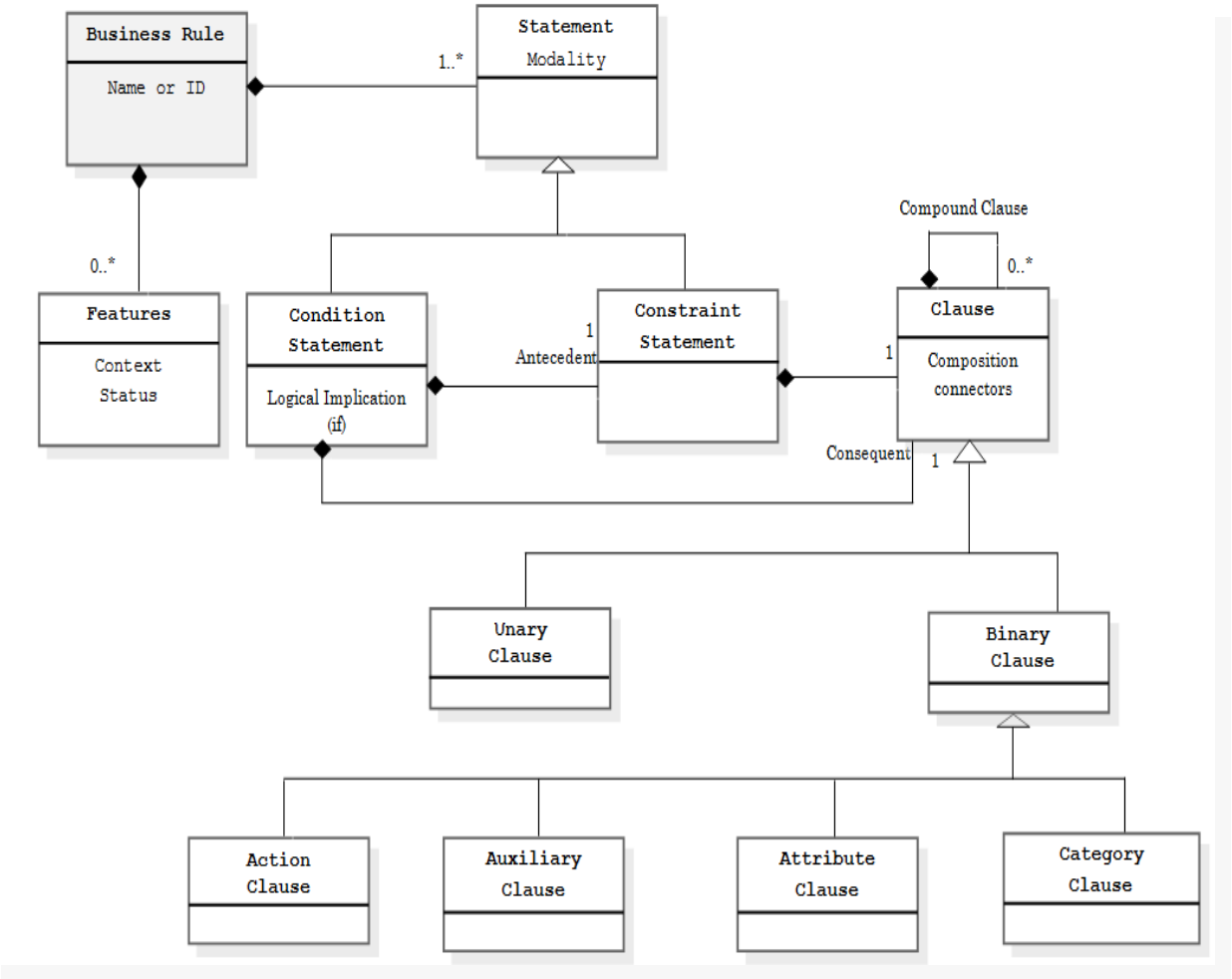


Figure 5.5 : Méta-modèle pour la représentation d'une règle métier dans RuleCNL

Comme le montre la figure 5.5 ci-dessus, une règle métier dans RuleCNL est composée d'une ou de plusieurs déclarations de la forme : «*Modality*» + «*Statement* ». Une règle métier est identifiée par son nom ou son identité (ID) et peut bien évidemment avoir quelques propriétés telles que son contexte, son statut (validé ou non) etc. Le «*Modality*» représente le concept de modalité présenté ci-dessus. Le «*Statement* » est une phrase déclarative qui régule la structure de la règle. Il peut être une contrainte (*Constraint Statement*) qui est une règle toujours vraie ou une condition (*Condition Statement*) qui est une règle vraie sous certaines conditions. Une règle métier peut être composée d'une clause simple (basée sur exactement un *Domain Verb*) ou d'une clause complexe. Une clause complexe est une clause composée d'au moins deux clauses simples. Cette composition de clauses simples se fait par des conjonctions de coordination (*and, or*), par des conjonctions de subordination et des pronoms relatifs (*if, that, who, which, etc.*). Une clause est toujours basée sur un *Domain Verb* qui est défini dans le vocabulaire. Par la suite, elle combine de nombreux *Rule Concepts* présentés à la figure 5.4 et des particules linguistiques (adjectifs, adverbes, etc.) afin de former une règle grammaticalement et sémantiquement correcte. Une clause peut être unaire ou binaire selon qu'elle est basée sur un *Domain Verb* unaire ou un *Domain Verb* binaire. Nous détaillons chacun des cas dans les sections suivantes.

a. Les clauses unaires

La figure 5.6 présente le méta-modèle des clauses unaires.

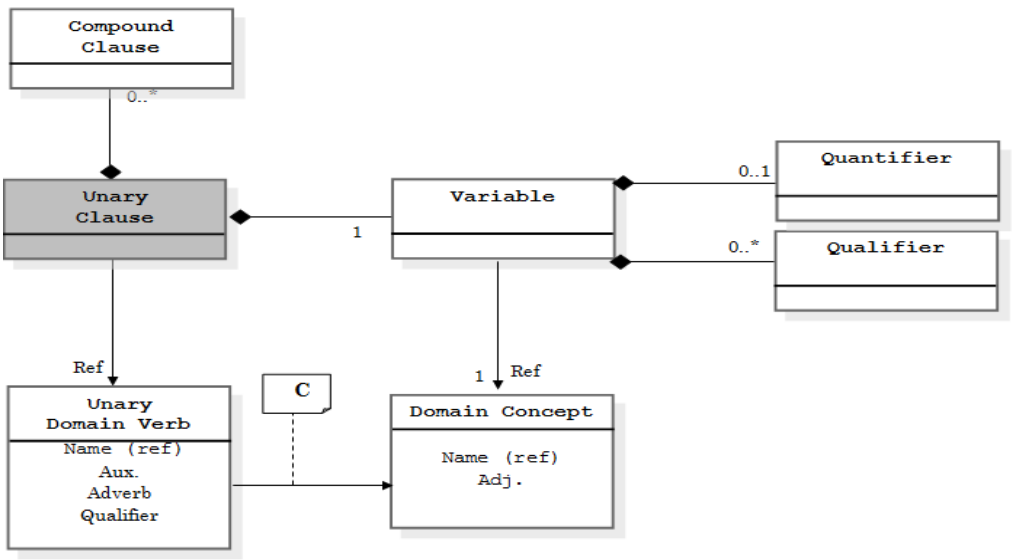


Figure 5.6 : Méta-modèle des clauses unaires

Comme le montre la figure 5.5, une clause unaire est basée sur un *Domain Verb* unaire et est composée d'une variable. Elle est éventuellement composée d'une ou de plusieurs autres clauses composées (nous décrivons les clauses composées plus bas). Une variable fait référence au *Domain Concept* qui remplit le rôle du *Domain Verb.* Une contrainte est définie (symbole « C » de la figure 5.5) afin de vérifier que l'élément (*Domain Concept* + *Domain Verb)* est bien défini dans le vocabulaire. La variable peut être quantifiée par des quantificateurs universels ou existentiels. Elle peut également être qualifiée par d'autres éléments descriptifs (adjectifs, etc.) afin de spécifier la règle avec suffisamment de précision. Les exemples suivants illustrent notre description :

(1) **It is obligatory that each** order *is shipped.*

(2)**It is obligatory that the** order "Number 2"*is shipped.*

(3) **It is obligatory that each** order **that** *is placed by* **a** gold customer *be shipped "rapidly".*

Ces exemples sont basés sur les faits : order *shipped* et order *is placed by* gold customer définis dans le vocabulaire.

Dans le cas de l'exemple (1), le *Domain Term* order n'est pas qualifié et dans ce cas, la règle sera appliquée à toutes ses instances.

Dans le cas de l'exemple (2), le *Domain Term* order est qualifié par une instanciation (instance de order „Number 2„).

Dans le cas de l'exemple (3) le *Domain Term* order est qualifié par une restriction faite à toutes les instances de order qui satisfont le fait : order *is placed by* gold customer. Également, le *Domain Verb* « *is shipped* » est qualifié par l'adverbe *"rapidly"* qui précise bien le sens du *Domain Verb*.

Comme nous avons mentionné plus haut, une clause (unaire ou binaire) peut éventuellement être composée d'une ou de plusieurs autres clauses (unaire ou binaire). Dans ce cas, la première clause est appelée clause principale (basée sur le fait principal). Un exemple est donné en (4) :

(4) **Il est obligatoire que chaque** employé *soit titulaire* **et** *possède* **un** compte bancaire.

Dans cet exemple, le fait principal est : employé *titulaire* et l'autre fait associé qui le compose est : employé *possède* compte bancaire. La composition ici est établie par la conjonction de coordination (*et*).

b. Les clauses binaires

Une clause binaire est basée sur *Domain Verb* binaire. De la même façon qu'un *Domain Verb* peut être subdivisé en « *Action Domain Verb* », « *Attribute Domain Verb* » et « *Category Domain Verb* », une clause binaire est aussi subdivisée en « *Action Clause* », « *Attribute Clause* » et « *Category Clause* ». Nous distinguons un quatrième type à savoir une « *Auxiliary Clause* » qui est basée sur des relations construites à partir des auxiliaires *être* et *avoir*. Les figures 5.6, 5.7, 5.8 et 5.9 présentent les méta-modèles correspondant respectivement à ces différentes clauses.

- ***Action Clause***

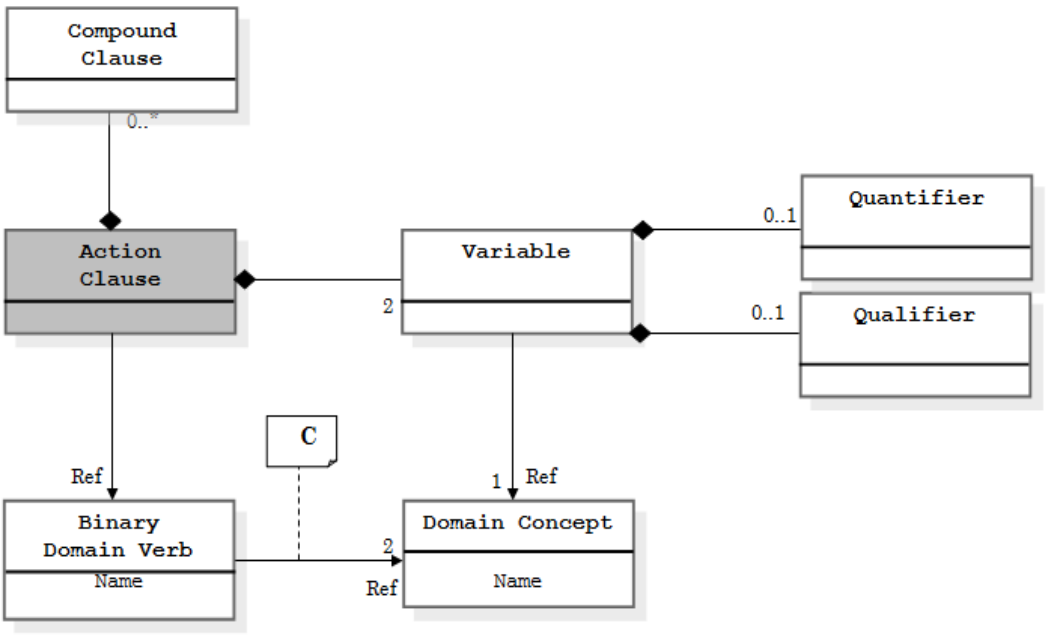


Figure 5.7 : Méta-modèle de la clause binaire *Action Clause*

Une *Action Clause* est basée sur un *Action Domain Verb* et est composée de deux variables. Elle est éventuellement composée d'une ou de plusieurs autres clauses. Chaque variable fait référence à un *Domain Concept* qui remplit un rôle dans la relation définie par le *Action*

Domain Verb. Une contrainte est également définie afin de vérifier que l'élément (*Domain Concept* (Role1) + *Action Domain Verb* + *Domain Concept* (Role2)*)* est bien défini dans le vocabulaire. Chaque variable peut être quantifiée par des quantificateurs universels ou existentiels. Elle peut également être qualifiée par d'autres éléments descriptifs afin de spécifier la règle avec suffisamment de précision comme dans le cas des clauses unaires décrites plus haut. Les exemples suivants illustrent ces propos :

(1)**It is obligatory that each** customer *place* **at least one** order.

(2)**It is necessary that the** customer *place* **an** order **if** the customer *holds* **a** bank account.

(3) **Il est obligatoire que chaque** client *possède* **au moins 1** compte bancaire **et** *soit majeur.*

- ***Attribute Clause***

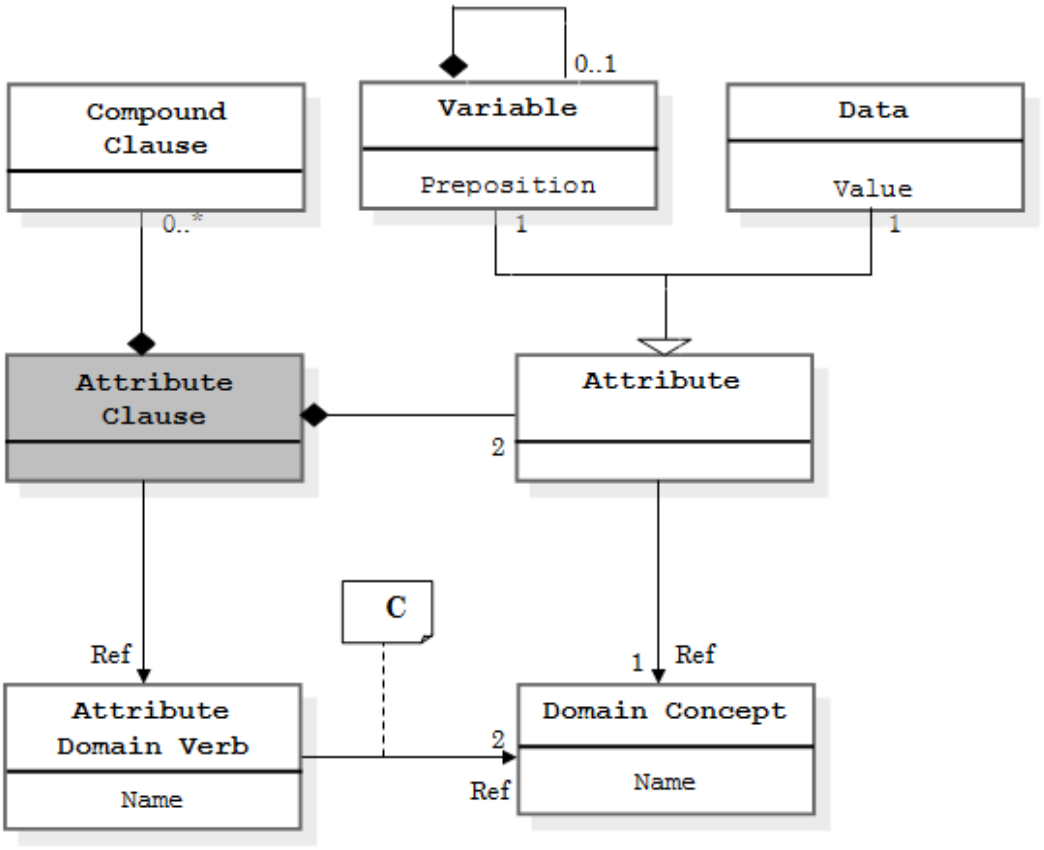


Figure 5.8 : Méta-modèle de la clause binaire *Attribute Clause*

Tout comme une *Action Clause*, une *Attribute Clause* est basée sur un *Attribute Domain Verb* et est composée de deux attributs. Elle est éventuellement composée d'une ou de plusieurs autres clauses. Un attribut est soit une variable, une propriété d'une variable (dans ce cas reliée avec la préposition *of*) ou une donnée. Dans le cas où l'attribut est une variable, il fait référence à un *Domain Concept* qui remplit un rôle dans la relation définie par le *Attribute Domain Verb*. Dans ce cas, une contrainte est définie afin de vérifier la consistance avec le vocabulaire. Elle peut être quantifiée par des quantificateurs universels ou existentiels. Elle peut également être qualifiée par d'autres éléments descriptifs afin de spécifier la règle avec suffisamment de précision. Les exemples ci-dessous illustrent quelques cas :

(1) **Il est nécessaire que la** période d'appréciation *soit supérieure ou égal à* **12 mois.**

(2)**It is necessary that the** age *of* **the** customer **who** *places* **the** order *be greater than* **18 years old.**

(3) **It is prohibited that** the outstanding balance *of the* account *be less than* **the** credit authorization threshold.

Category Clause

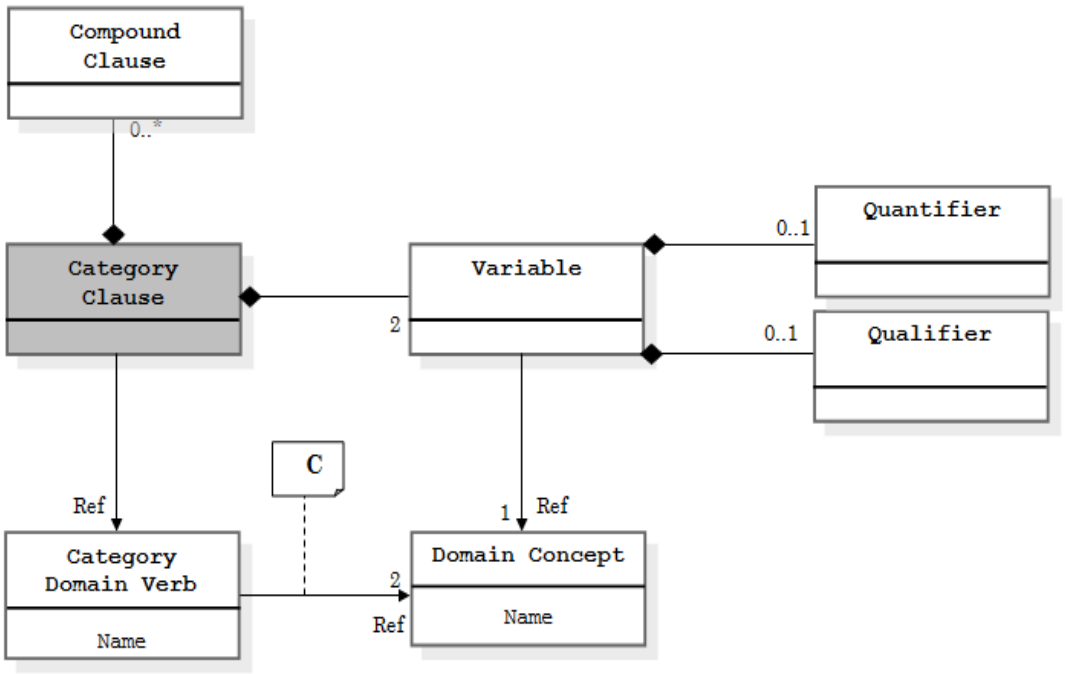


Figure 5.9 : Méta-modèle de la clause binaire *Category Clause*

Tout comme une *Action Clause*, une *Category Clause* est basée sur un *Category Domain Verb* et est composée de deux variables. Elle est éventuellement composée d'une ou de plusieurs autres clauses. Chaque variable fait référence à un *Domain Concept* qui remplit un rôle dans la relation définie par le *Category Domain Verb.* Une contrainte est également définie afin de vérifier que l'élément (*Domain Concept* (Role1) + *Category Domain Verb* + *Domain Concept* (Role2)*)* est bien défini dans le vocabulaire. Chaque variable peut être quantifiée par des quantificateurs universels ou existentiels. Elle peut également être qualifiée par d'autres éléments descriptifs afin de spécifier la règle avec suffisamment de précision comme dans le cas des clauses unaires décrites plus haut. L'exemple ci-dessous présente une *Category Clause.*

Il est nécessaire que le client *soit un* client prioritaire **si le** client *passe* **au moins 10** commandes.

- ***Auxiliary Clause***

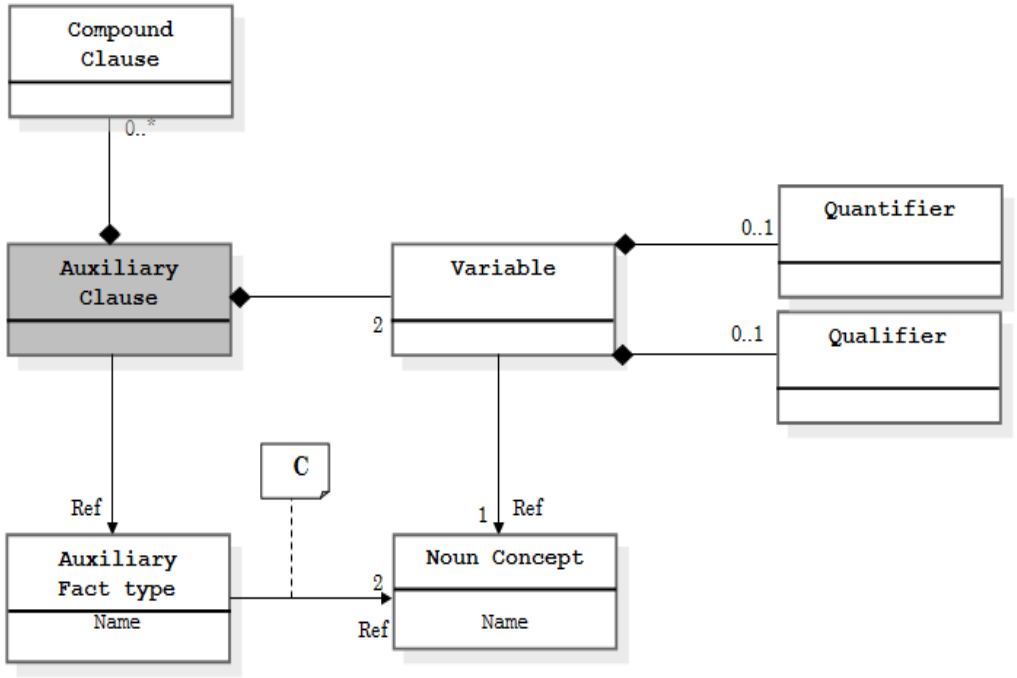


Figure 5.10 : Méta-modèle de la clause binaire *Auxiliary Clause*

Une *Auxiliary Clause* est basée sur le verbe auxiliaire (*être*) et est composée de deux variables. Elle permet principalement de représenter les ensembles. Par exemple, si le terme métier A est dans l'ensemble {A1, A2, A3}, ceci peut être représenté par : A *est* **un** A1 **ou** *est* **un** A2 ou *est* **un** A3. Tout comme une *Action Clause,* elle est composée de deux variables et est éventuellement composée d'une ou de plusieurs autres clauses. Chaque variable peut être quantifiée par des quantificateurs universels ou existentiels. Elle peut également être qualifiée par d'autres éléments descriptifs afin de spécifier la règle avec suffisamment de précision comme dans le cas des clauses unaires décrites plus haut. Nous avons ci-dessous un exemple.

Il est nécessaire que le congé *soit compté comme* période d'activité **si le** type de congé *est* **un** congé annuel **ou** *est* **un** congé de maternité **ou** *est* **un** congé de paternité **ou** *est* **un** congé de formation professionnelle.

- ***Compound Clause***

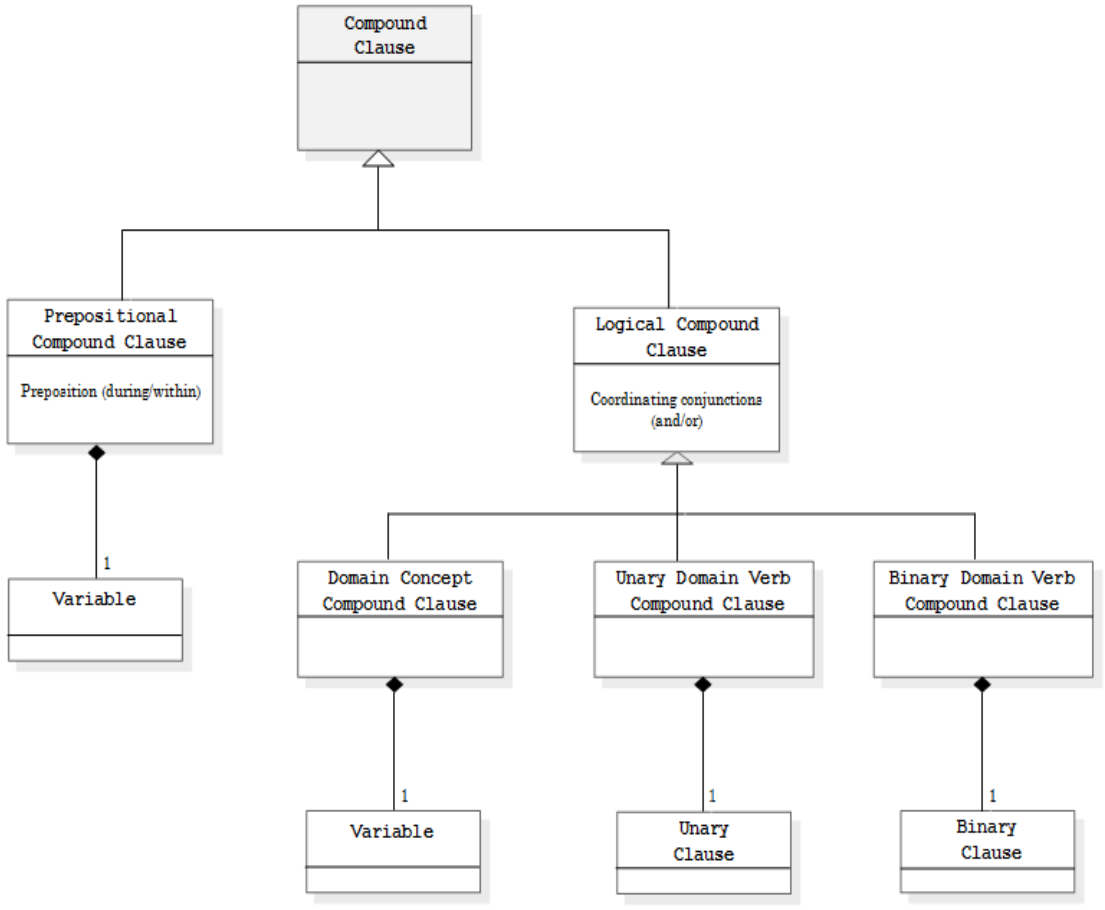


Figure 5.11 : Méta-modèle de la clause *Compound Clause*

Une *Compound Clause* permet tout simplement de relier par l'intermédiaire des connecteurs une ou plusieurs autres clauses à une clause principale. Nous distinguons deux types de clauses composées :

- Les clauses composées prépositionnelles qui sont des clauses qui relient la clause principale à d'autres variables par l'intermédiaire des prépositions.

Par exemple : **It is obligatory that a** teacher *give* course **in a** class **during a** date.
Dans cet exemple, le fait principal est (teacher *gives* course).

- Les clauses composées logiques qui sont des clauses qui relient la clause principale à d'autres clauses par l'intermédiaire des conjonctions de coordination. Les clauses composées logiques peuvent à leur tour être catégorisées en *Domain Concept Compound Clause, Unary Domain Verb Compound Clause, et Unary Domain Verb Compound Clause.*

Une *Domain Concept Compound Clause* est une clause qui relie la clause principale à une variable comme illustré dans l'exemple ci-dessous.

Il est nécessaire que le congé *soit compté comme* période d'activité **si le** type de congé *est* **un** congé annuel **ou un** congé de maternité.

Dans cet exemple, le fait principal est (type de congé *est* congé annuel) et la variable reliée à la clause principale est (**un** congé de maternité).

Une *Unary Domain Verb Compound Clause* est une clause qui relie la clause principale à une clause unaire comme illustré dans l'exemple ci-dessous.

Il est obligatoire que le congé *soit compté comme* période d'activité **si le** congé *est accordé* **et** *est rémunéré.*

Dans cet exemple, le fait principal est (congé *accordé*) et le fait unaire relié à la clause principale est (congé *rémunéré*).

Une *Binary Domain Verb Compound Clause* est une clause qui relie la clause principale à une clause binaire comme illustré dans l'exemple ci-dessous.

Il est obligatoire que le congé d'arrêt maladie *soit compté comme* période d'activité **si l'**employé *est titulaire* **et** *possède* **un** certificat médical.

Dans cet exemple, le fait principal est (employé *titulaire*) et le fait binaire relié à la clause principale est (employé *possède* certificat médical).

Quantifier

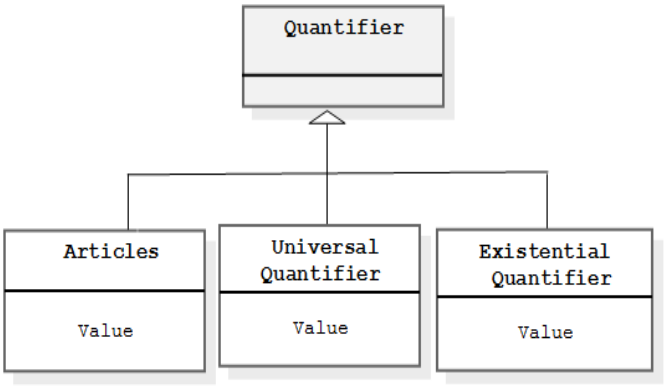


Figure 5.12 : *Quantifier*

Un *Quantifier* est soit un article indéfini ou défini (*a, an, the, etc.*), soit une quantification universelle (chaque, etc.) soit une quantification existentielle pouvant être combiné avec un *Domain Concept* pour former une variable (groupe ou syntagme nominal).

Exemple : **Chaque** client ; **au moins un** client ; **exactement un** client ; etc.

Qualifier

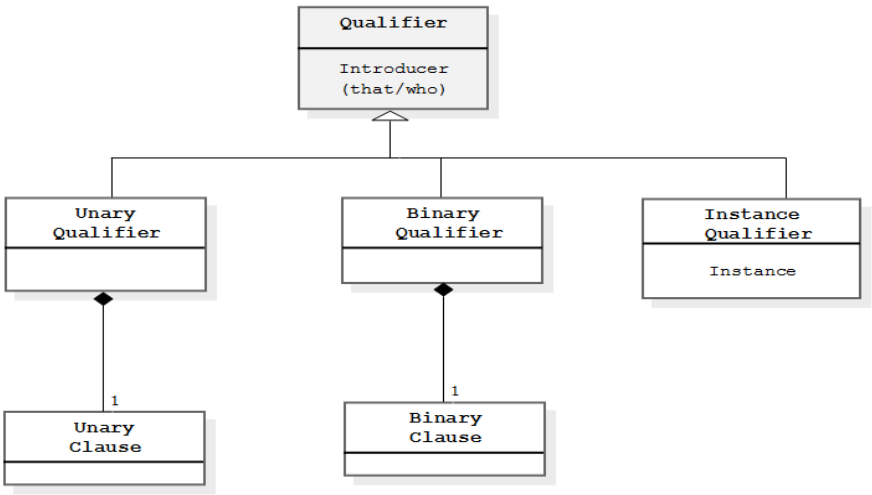


Figure 5.13 : *Qualifier*

Un *qualifier* est un élément descriptif permettant de qualifier un terme métier afin d'apporter plus de précision ou lui imposer certaines restrictions.

Un *qualifier* peut être une instanciation d'un *Domain Term (*: …**le** client „Dupont"…). Il peut être une restriction introduite par un pronom relatif (*who, that*). La restriction peut être basée soit sur une clause unaire (…client **qui** *soit prioritaire*…), soit sur une clause binaire (…client **qui** *possède* **un** compte bancaire…).

5.3.3 Grammaire de RuleCNL

RuleCNL est un langage contrôlé naturel c'est-à-dire un sous-ensemble du langage naturel. Ainsi, sa syntaxe concrète définie à partir du méta-modèle décrit ci-dessus doit être textuelle. Elle doit être décrite à partir d'une grammaire qui définie les structures syntaxiques des règles métier.

5.3.3.1 Les grammaires des langages contrôlés

La grammaire représente un élément très important dans la conception des langages contrôlés car elle constitue le principal moyen de définir formellement la syntaxe du langage. La source la plus courante d'informations syntaxiques qui peut être utilisée pour analyser les éléments d'entrées est la grammaire du langage (Denaux, 2013). Puisque les langages contrôlés sont basés sur des langages naturels (langages de base), leurs grammaires partagent de nombreuses propriétés avec les grammaires des langages naturels. Par exemple, elles décrivent certains mêmes mots et classes de mots et utilisent les mêmes structures syntaxiques comme les phrases, les groupes nominaux, les groupes verbaux, et les clauses relatives. La grammaire d'un langage naturel est basée sur le lexique du langage qui comprend tous les mots décrits dans un dictionnaire général avec d'autres informations possibles (morphologie, partie du discours, inflexion, etc.). À cause des ambigüités du langage naturel, l'interprétation formelle de la grammaire n'est pas toujours facile et fiable.

Cependant, l'objectif des langages contrôlés est de s'assurer que le texte rédigé dans le dit langage puisse être interprété ou converti de façon déterministe dans une logique formelle particulière en utilisant un algorithme approprié. Pour ces raisons, les langages contrôlés orientés-machines doivent avoir des grammaires formelles accompagnées des parseurs qui convertissent ses phrases vers la logique formelle. La logique du premier ordre et ses extensions sont les plus expressives, tandis que les clauses de Horn et les logiques

descriptives sont des sous-ensembles de la logique du premier ordre qui permettent une implémentation efficace. Ainsi, les langages contrôlés ont des exigences concernant leurs grammaires qui diffèrent de celles des langages naturels et aussi de celles des langages informatiques.

La spécification d'une grammaire formelle (grammaire hors-contexte, *definite clause grammar*, grammaire régulière, etc.) pour définir un langage contrôlé pour les règles métier impose un compromis entre l'implémentabilité et l'expressivité. L'implémentabilité a une incidence sur la facilité de développement du langage lui-même et les outils support. L'expressivité permet la facilité d'utilisation pour les experts métier qui spécifient ces règles métier. La plupart des experts métier ne connaissent toujours pas les règles de production ou de dérivation spécifiques à la grammaire. Ils doivent seulement être au courant des éléments de haut niveau tels que les concepts et les mots-clés. Les grammaires formelles sont à la fois utiles pour des outils et aux développeurs des langages. Elles définissent sans ambiguïtés les règles du langage et comment ces règles sont spécifiées. Pour les utilisateurs finaux d'un langage contrôlé, la grammaire peut être un obstacle quand ils ne sont pas au courant de toutes les règles de la grammaire. Ainsi, afin d'être facilement utilisable, des éditeurs prédictifs avec les caractéristiques de *look-ahead* c'est-à-dire la possibilité de déduire avec un effort raisonnable les mots pouvant suivre un texte partiel devraient être associés à ces langages efficacement.

De façon générale, les grammaires formelles des langages contrôlés doivent entre autre être concrètes, déclaratives et facilement implémentables avec des fonctionnalités de *look-ahead* (Kuhn, 2010).

❖ La caractéristique « concrète » des grammaires formelles est une exigence évidente. En raison de leurs propriétés d'être pratiques et orientés-machines, ces grammaires doivent être concrètes car les grammaires concrètes sont entièrement formalisables et peuvent être lues et interprétées par des programmes, par opposition aux grammaires abstraites qui peuvent être informels et ne peuvent être traitées automatiquement sans un travail considérable (Kuhn, 2010).

❖ Comme deuxième exigence, ces grammaires devraient être déclaratives afin de faciliter leur usage par les différents types d'outils. Une grammaire est déclarative si elle est définie de façon à être indépendante de tout algorithme concret ou de toute implémentation.

Les grammaires déclaratives ont l'avantage qu'elles peuvent être complètement séparées des parseurs qui les traitent. Cela rend ces grammaires flexibles car elles peuvent être utilisées par différents programmes et parseurs avec aussi la possibilité d'avoir différents parseurs pour le même langage. Un autre avantage des grammaires déclaratives est qu'elles peuvent facilement être partagées et réutilisées (Kuhn, 2010).

❖ Enfin, les grammaires formelles devraient être faciles à implémenter dans différents langages de programmation. En conséquence, la notation de la grammaire devrait être neutre par rapport au paradigme de programmation de son parseur. L'implementabilité est motivée par le fait que la facilité d'utilisation d'un langage contrôlé dépend fortement de la bonne convivialité des interfaces utilisateurs, comme les éditeurs prédictifs. Les éditeurs prédictifs exigent la disponibilité des fonctionnalités de *look-ahead*, c'est-à-dire la possibilité de savoir comment un texte partiel peut être poursuivi. Pour cette raison, les langages contrôlés doivent être définis de façon à permettre l'implémentation efficace de ces caractéristiques de *look-ahead*. Il est souhaitable que le parseur d'un langage contrôlé soit implémenté dans le même langage de programmation que les composants de l'interface utilisateur de l'outil support (Kuhn, 2010).

5.3.3.2 Formalisation de la grammaire de RuleCNL

La grammaire de RuleCNL est une grammaire hors-contexte ou *Context-Free Grammar* (CFG) car elle remplie les exigences citées plus haut. Les grammaires hors-contextes peuvent être décrites de façon déclarative et simple en utilisant la notation *Backus Naur Form* (BNF) ou *Extended BNF* (EBNF)[30] pouvant être implémentée. Nous décrivons ainsi la grammaire de RuleCNL en utilisant la notation EBNF. La recherche autour des grammaires hors-contextes est relativement mature et les outils supports comme Yacc (Jonhson, 1974), *ANother Tool for Language Recognition*[31] (ANTLR) (Parr et Quong, 1995) développés à l'origine pour écrire les compilateurs, sont de nos jours bien maitrisés. Cette maturité se traduit par la possibilité pour ces outils de générer automatiquement des parseurs à partir de la spécification de la grammaire.

[30] BNF - http://www.garshol.priv.no/download/text/bnf.html

[31] ANTLR - http://www.antlr.org

Il faut cependant noter que les grammaires hors-contextes ont quelques inconvénients car elles peuvent générer des phrases syntaxiquement ambiguës (phrases avec plus d'une dérivation). Ainsi, il revient aux concepteurs du langage de s'efforcer à éviter ces ambiguïtés parce qu'elles rendent le langage plus difficile à analyser. Certaines contraintes sémantiques contextuelles ne peuvent pas être définies de manière déclarative dans la grammaire, aussi elles nécessitent d'être gérées de façon procédurale par le concepteur du langage.

La grammaire de RuleCNL est composée d'un ensemble de règles de réécriture permettant de contrôler la syntaxe des déclarations de règles métier. Dans la notation EBNF, La barre verticale « | » représente la disjonction et la virgule « , » représente la conjonction. La parenthèse suivie su signe « + » « ( )$^{+}$ » signifie qu'au moins une occurrence de l'élément entre parenthèse existe c'est-à-dire un ou plusieurs. La parenthèse suivie du signe « * » « ( )* » signifie qu'au moins zéro occurrence de l'élément entre parenthèse existe. La parenthèse suivie su signe « ? » « ( ) ? » signifie que l'élément entre parenthèse est optionnel. Le méta-symbole « : » sépare la partie gauche de la règle qui est toujours un symbole non terminal de la partie droite. Tous les symboles non terminaux doivent apparaitre à la partie gauche d'au moins une règle de réécriture. Les symboles non terminaux commencent nécessairement par une lettre majuscule et les symboles terminaux sont écrits entre guillemets. Le méta-symbole « ref:: » permet le référencement d'un élément du vocabulaire.

La figure 5.13 montre l'architecture de mise en œuvre de la grammaire à partir du méta-modèle ainsi que le parseur de RuleCNL.

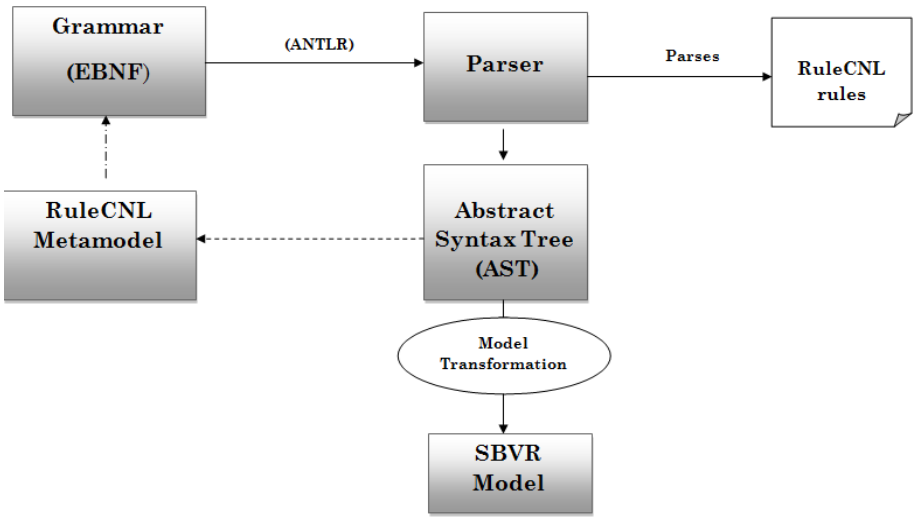


Figure 5.14 : Spécification de la grammaire de RuleCNL à partir de son méta-modèle

Comme le montre la figure 5.13 ci-dessus, la grammaire de RuleCNL est décrite à partir de son méta-modèle en utilisant la notation EBNF. Un extrait de cette grammaire est présenté à la figure 5.14. L'outil ANTLR est utilisé pour générer le parseur du langage à partir de la grammaire. Ce parseur est utilisé par les éditeurs pour vérifier la syntaxe des règles métier. Il est aussi à la base d'autres fonctionnalités telles que l'auto-complétion automatique, la coloration syntaxique, la gestion des erreurs et la validation des règles métier. À la suite de l'analyse d'une règle par le parseur, l'arbre syntaxique ou AST (*Abstract Syntax Tree*) est généré. Nous l'adaptons grâce à des algorithmes appropriés pour qu'il soit bel et bien conforme au méta-modèle de RuleCNL car il constitue le point d'entrée pour les transformations automatiques vers des modèles sémantiques SBVR.

```
BusinessRule:  RuleID,  RuleDescription, Point;
RuleDescription:  RuleModality, BusinessRuleStatement;
RuleID:  STRING;
RuleModality :   ListRuleModality ;
BusinessRuleStatement: Clause, (LogicalORaction)?;
LogicalORaction: (LogicalClause)+ |ActionRule ;
LogicalClause : LogicalOperator, Clause;
LogicalOperator:   ListLogicalOperator;
ActionRule :  OperatorIf, ConstraintRule;
ConstraintRule: Clause, (LogicalClauseIf)* ;
LogicalClauseIf : LogicalOperatorIf, Clause;
LogicalOperatorIf:   ListLogicalOperatorIf ;
OperatorIf : ListOperatorIf;
Clause  :  ActionBinaryClause | UnaryClause | AttributeBinaryClause | CategoryBinaryClause | AuxilaryBinaryClause ;
ActionBinaryClause : Variable,  (Auxiliaire)?, (ref::BinaryVerb), Variable, (CompoundExpression)*;
AttributeBinaryClause : Attribute, Auxiliaire, OperatorVerb,  Preposition,  Attribute, (CompoundExpression)*;
CategoryBinaryClause :  Variable, Auxiliaire, CategoryVerb, Variable, (CompoundExpression)*;
AuxilaryBinaryClause : Variable, AuxiliaireVariable, (CompoundExpression)*;
UnaryClause :  Variable, Auxiliaire, (adverb = STRING)?, (ref::UnaryVerb), (qualifier = STRING)?, (CompoundExpression)*;
Variable:  ( Quantifier)?, (adverb = STRING)?,(ref::NounConcept), (Qualifier)*;
Qualifier :    UnaryFactQualifier | BinaryFactQualifier | InstantiationQualifier ;
CompoundExpression : LogicalOperator CompoundVerbConcept;
 CompoundVerbConcept:   UnaryCompoundVerbConcept |ActionCompoundVerbConcept |AttributeCompoundVerbConcept |
                                                  CategoryCompoundVerbConcept|AuxilaryCompoundVerbConcept ;
UnaryCompoundVerbConcept :     ( Auxiliaire)?,(adverb = STRING)?,(ref::UnaryVerb), (qualifier = STRING)?;
ActionCompoundVerbConcept :    (Auxiliaire)?, (ref::BinaryVerb), Variable;
AttributeCompoundVerbConcept :  (Auxiliaire)?, OperatorVerb, Preposition,  (Quantifier)?, NounConceptORValue;
CategoryCompoundVerbConcept :  (Auxiliaire)?, CategoryVerb, Variable;
```

```
AuxilaryCompoundVerbConcept :  (Auxiliaire)?, Variable ;
BinaryFactQualifier:  ActionBinaryFactQualifier |AttributeBinaryFactQualifier|CategoryBinaryFactQualifier ;
ActionBinaryFactQualifier: Introducer, (Auxiliaire)?, (ref::BinaryVerb), (Quantifier)?, (ref::NounConcept);
AttributeBinaryFactQualifier: Introducer, Auxiliaire,  OperatorVerb,  Preposition, (Quantifier)?, NounConceptORValue;
CategoryBinaryFactQualifier: Introducer, Auxiliaire, CategoryVerb, (Quantifier)?, (ref::NounConcept) ;
NounConceptORValue : (ref::NounConcept) | ValueAttribute;
UnaryFactQualifier:  Introducer, Auxiliaire, (adverb = STRING)?, (ref::UnaryVerb),(qualifier = STRING)?;
InstantiationQualifier : STRING;
Auxiliaire : ListAuxiliaire ;
Quantifier :  UniversalQuantifier | ExistentialQuantifier ;
UniversalQuantifier : ListUniversalQuantifier;
ExistentialQuantifier :  ListExistentialQuantifier, INT ;
Preposition : ListPreposition];
Introducer : ListIntroducer ;
Attribute:    VariableAttribute | ConceptAttribute | ValueAttribute ;
ConceptAttribute:   Variable, Prep, Variable;
VariableAttribute:   Variable;
ValueAttribute :  INT,(STRING)?) | STRING ;
Prep : ListPrep;
OperatorVerb :PositiveOperatorVerb | NegativeOperatorVerb ;
PositiveOperatorVerb : ListPositiveOperatorVerb ;
NegativeOperatorVerb : ListNegativeOperatorVerb ;;
CategoryVerb : ListCategoryVerb;
Point : ".";
STRING :   ('a'..'z'|'A'..'Z'|'_'|'é'|'è'|'ê'|'ç'|'à'|'ë')('a'..'z'|'A'..'Z'|'_'|'-'|'0'..'9'|'é'|'è'|'ê'|'ç'|'à'|'ë')*;
```

Figure 5.15 : Extrait de la grammaire de RuleCNL en notation EBNF

5.4 Sémantique d'exécution de RuleCNL

Comme nous avons déjà mentionné plus haut, la définition d'un langage peut être considérée comme une activité décomposée en trois phases : la définition de la syntaxe, la définition du domaine sémantique et la correspondance entre la syntaxe et le domaine sémantique. Cette dernière phase est considérée comme la sémantique d'exécution du langage. Dans la section précédente, nous avons présenté la syntaxe de RuleCNL. Son domaine sémantique est constitué des formulations sémantiques de SBVR décrites en détail à la Section 3.5. Dans cette section, nous présentons la sémantique d'exécution de RuleCNL. Elle est définie par des transformations automatiques des règles métier exprimées en RuleCNL vers des formulations sémantiques SBVR (OMG, 2008a). L'intention derrière les formulations sémantiques de SBVR est de pouvoir capturer formellement les *Domain Concepts*, les *Domain Verbs* et les règles métier exprimées en RuleCNL. Les déclarations formelles de règles métier peuvent

ensuite être transformées en formulations logiques pouvant être lues par des outils logiciels. Les formulations logiques qui sont des sous-ensembles des formulations sémantiques constituent des déclarations des faits en logique formelle de premier ordre de la formalisation des règles métier. Ces déclarations des faits sont exprimées en XML suivant le standard XMI et conforme au schéma XML du méta-model SBVR.

Rappelons cependant que le méta-modèle SBVR fait parti intégrante de l'architecture MDA et appartient à la couche CIM. Toutefois, le problème de l'opérationnalisation (transformations vers des modèles PIM et PSM) de ces modèles sémantiques SBVR dans les SI reste de nos jours un grand challenge. Il est clair que le standard SBVR à la base ne vise pas à définir un langage à partir duquel on pourrait générer automatiquement des composants logiciels exécutables. Les modèles SBVR sont de nature déclarative, mettant l'accent sur le « Quoi » d'un SI, plutôt que de définir le «Comment» de son implémentation. Cependant, les fondements logiques des formulations sémantiques des règles métier constituent ainsi un point d'entrée vers la modélisation en langage naturel. Depuis la publication de SBVR en Janvier 2008, plusieurs recherches sont menées afin d'aller des modèles sémantiques de SBVR vers des diagrammes de classes UML (Raj et al., 2008 ; Kleiner, 2009 ; Afreen et Bajwa, 2011), vers le langage de requête SQL dans le but de valider la consistance d'une base de données (Moschoyiannis et al., 2010 ; Marinos et Krause, 2009), vers le langage d'expression de contraintes OCL (Bajwa et al., 2010), vers des langages de web sémantique comme R2ML (Demuth et Liebau, 2007).

La sémantique d'exécution de RuleCNL est basée sur un modèle de transformation de modèles axé sur la modélisation comme décrit à la Section 3.4.2.1.

La première étape consiste en la mise en correspondance des concepts du méta modèle de RuleCNL (décrit à la Section 5.3) et celui de SBVR pour former le méta-modèle de transformation. Cette mise en correspondance se fait à deux niveaux : la mise en correspondance du vocabulaire de RuleCNL et le vocabulaire de SBVR (cf. tableau 5.1) ; la mise en correspondance des concepts de RuleCNL avec les concepts des formulations sémantiques de SBVR (cf. tableau 5.2).

RuleCNL Vocabulary	SBVR Vocabulary
Domain Term	*Noun Concept*
Domain Name	*Individual Concept*
Unary Domain Verb	*Characteristic Fact type*
Binary Domain Verb	*Associative binary Fact type*
Action Domain Verb	*Associative Fact type*
Attribute Domain Verb	*Is_Property_of Fact type*
Category Domain Verb	*Categorization Fact type*

Tableau 5.1 : Mise en correspondance du vocabulaire de RuleCNL et celui de SBVR

La deuxième étape est l'expression des règles de transformation qui permettent de mapper chaque élément du vocabulaire de RuleCNL avec le pattern XML correspondant du schéma XML du vocabulaire de SBVR (*SBVR XML Schema*) défini à la clause 13.3 de (OMG, 2008a). Les exemples de ces patterns XML pour les *Noun Concepts*, *Individual Concepts*, *Characteristic Fact types*, *Associative Binary Fact types* sont présentés respectivement sur les figures 5. 15, 5. 16, 5. 17 et 5. 18.

RuleCNL Rule Concepts	SBVR Semantic formulations
Modal Concept *Obligation* *Necessity* *Permission* *Possibility*	***Modal Formulation*** *Obligation formulation* *Necessity formulation* *Permissibility formulation* *Possibility formulation*
Quantifier Concept *Definite article* *Indefinite article* *Universal quantification (each, all, etc.)* *Existential quantification (at least, more than, etc.)*	***Quantification Formulation*** *Existential quantification* *Universal quantification* *Universal quantification* *Existential quantification*
Connector Concept *Coordinating conjunction and* *Coordinating conjunction or*	***Binary Logical Operation*** *Logical conjunction* *Inclusive Logical disjunction*
Prepositional Concept	***Projection Formulation***
Linguistic Concept *Subordinating conjunction if* *Relative pronouns that, who, which*	***Binary Logical Operation*** *Logical implication* *Projection formulation, instantiation formulation*
Domain Verb	***Atomic Formulation***

Tableau 5.2 : Mise en correspondance des concepts de RuleCNL et des formulations sémantiques de SBVR

example term

```
<sbvr:term xmi:id="exampleTerm" signifier="et-s" meaning="meaning"/>
<sbvr:objectType xmi:id="meaning"/>
<sbvr:text xmi:id="et-s" value="example term"/>
<sbvr:thingIsInSet set="vocabulary" thing="exampleTerm"/>
<sbvr:designationIsInNamespace designation="exampleTerm" namespace="vocabularyNamespace"/>
```

If there is no "See:" caption, then the following is included:

```
<sbvr:preferredDesignation xmi:id="exampleTermPreferred"/>
<sbvr:thing1IsThing2 thing1="exampleTermPreferred" thing2="exampleTerm"/>
```

Figure 5.16 : Pattern XML pour un *Noun Concept* (OMG, 2008a)

Example Name

```
<sbvr:name xmi:id="exampleName" signifier="en-s" meaning="meaning"/>
<sbvr:individualConcept xmi:id="meaning"/>
<sbvr:text xmi:id="en-s" value="Example Name"/>
<sbvr:thingIsInSet set="vocabulary" thing="exampleName"/>
<sbvr:designationIsInNamespace designation="exampleName" namespace="vocabularyNamespace"/>
```

If there is no "See:" caption, then the following is included:

```
<sbvr:preferredDesignation xmi:id= "exampleNamePreferred"/>
<sbvr:thing1IsThing2 thing1="exampleNamePreferred" thing2="exampleName"/>
```

Figure 5.17 : Pattern XML pour un *Individual Concept* (OMG, 2008a)

example *is seen*

```
<sbvr:sententialForm xmi:id="exampleIsSeen" expression="eis-e" meaning="meaning" placeholder="eis-p"/>
<sbvr:factSymbol xmi:id="example.isSeen" signifier="isSeen-s" meaning="meaning"/>
<sbvr:characteristic xmi:id="meaning" role="eis-r"/>
<sbvr:factTypeFormDemonstratesDesignation factTypeForm="exampleIsSeen" designation="example.isSeen"/>
<sbvr:text xmi:id="eis-e" value="example is seen"/>
<sbvr:text xmi:id="isSeen-s" value="is seen"/>
<sbvr:placeholder xmi:id="eis-p" expression="example-s" startingCharacterPosition="i1" meaning="eis-r"/>
<sbvr:placeholderUsesDesignation placeholder="eis-p" designation="example"/>
<sbvr:positiveInteger xmi:id="i1" value="1"/>
<sbvr:factTypeRole xmi:id="eis-r"/>
<sbvr:roleRangesOverObjectType role="eis-r" objectType="example-concept"/>
<sbvr:thingIsInSet set="vocabulary" thing="exampleIsSeen"/>
<sbvr:thingIsInSet set="vocabulary" thing="example.isSeen"/>
<sbvr:factTypeFormIsInNamespace factTypeForm="exampleIsSeen" namespace="vocabularyNamespace"/>
<sbvr:attributiveNamespaceIsWithinVocabularyNamespace attributiveNamespace="example-ans"
    vocabularyNamespace="vocabularyNamespace"/>
<sbvr:attributiveNamespace xmi:id="example-ans"/>
<sbvr:attributiveNamespaceIsForSubjectConcept attributiveNamespace="example-ans"
    subjectConcept="example-concept"/>
<sbvr:designationIsInNamespace designation="example.isSeen" namespace="example-ans"/>
```

Figure 5.18 : Pattern XML pour un *Characteristic Fact type* (OMG, 2008a)

example$_1$ *follows* example$_2$

```
<sbvr:sententialForm xmi:id="example1FollowsExample2" expression="efe-e" meaning="meaning" placeholder="efe-p1 efe-p2"/>
<sbvr:factSymbol xmi:id="efe-follows" signifier="follows-s" meaning="meaning"/>
<sbvr:binaryFactType xmi:id="meaning" role="efe-r1 efe-r2"/>
<sbvr:factTypeFormDemonstratesDesignation factTypeForm="example1FollowsExample2" designation="efe-follows"/>
<sbvr:text xmi:id="efe-e" value="example1 follows example2"/>
<sbvr:text xmi:id="follows-s" value="follows"/>
<sbvr:text xmi:id="example1-s" value="example1"/>
<sbvr:text xmi:id="example2-s" value="example2"/>
<sbvr:placeholder xmi:id="efe-p1" expression="example1-s" startingCharacterPosition="i1" meaning="efe-r1"/>
<sbvr:placeholder xmi:id="efe-p2" expression="example2-s" startingCharacterPosition="i18" meaning="efe-r2"/>
<sbvr:placeholderUsesDesignation placeholder="efe-p1" designation="example"/>
<sbvr:placeholderUsesDesignation placeholder="efe-p2" designation="example"/>
<sbvr:positiveInteger xmi:id="i1" value="1"/>
<sbvr:positiveInteger xmi:id="i18" value="18"/>
<sbvr:factTypeRole xmi:id="efe-r1"/>
<sbvr:factTypeRole xmi:id="efe-r2"/>
<sbvr:roleRangesOverObjectType role="efe-r1" objectType="example-concept"/>
<sbvr:roleRangesOverObjectType role="efe-r2" objectType="example-concept"/>
<sbvr:thingIsInSet set="vocabulary" thing=" example1FollowsExample2"/>
<sbvr:thingIsInSet set="vocabulary" thing=" efe-follows"/>
<sbvr:factTypeFormIsInNamespace factTypeForm="example1FollowsExample2" namespace="vocabularyNamespace"/>
```

Figure 5.19 : Pattern XML pour un *Associative Binary Fact type* (OMG, 2008a)

La troisième et dernière étape consiste à exécuter les règles de transformation avec un programme approprié. Ce programme prend en entrée un modèle RuleCNL (qui est constitué d'une ou de plusieurs règles métier exprimées dans la syntaxe de RuleCNL ainsi que le vocabulaire associé) et le méta-modèle de transformation afin de générer en sortie le modèle sémantique SBVR. Il s'agit de deux fichiers XML (le fichier du vocabulaire et le fichier des règles métier) conforme au Schéma XML du méta-modèle SBVR. Le fait de séparer ces deux fichiers est un principe clé du standard SBVR qui consiste à séparer le vocabulaire métier des règles métier. Ainsi, le vocabulaire métier pourra être utilisé à d'autres fins indépendamment des règles métier (par exemple dans les processus métier, etc.). Ce modèle sémantique SBVR peut de ce fait être partagé non seulement entre les entreprises et les outils logiciels, mais être également transformé dans un contexte MDA vers d'autres modèles PIM et PSM dans le contexte de modélisation des systèmes d'informations.

La figure 5.19 présente un exemple de workflow de transformation à partir d'une seule règle exprimée en RuleCNL vers des structures sémantiques SBVR. Le vocabulaire utilisé dans la règle est présenté à la figure 5.20. Dans la section suivante, nous présentons l'outil support de RuleCNL.

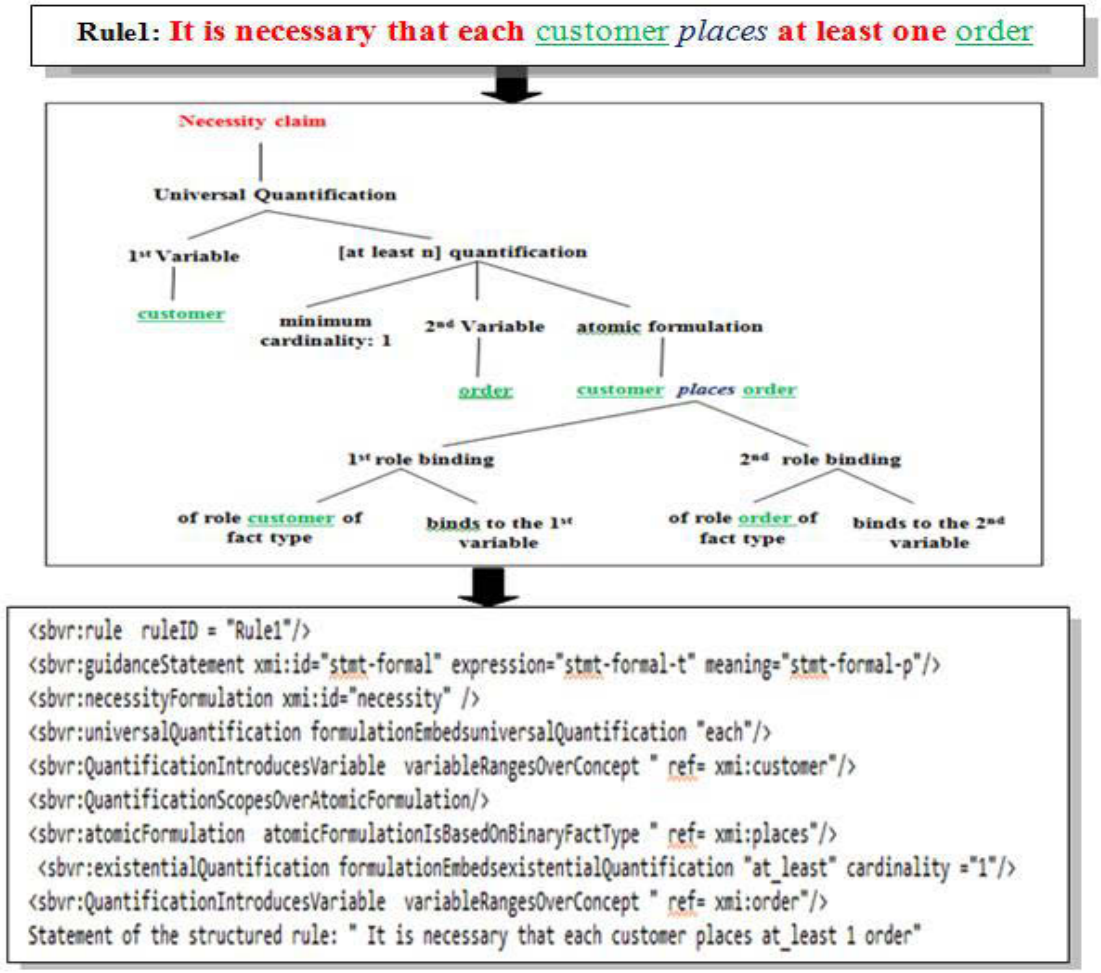


Figure 5.20 : Workflow de transformation à partir d"une règle exprimée en RuleCNL

```
<sbvr:term xmi:id="customer" signifier="customer-t" meaning="customer-c"/>
<sbvr:objectType xmi:id="customer-c"/>
<sbvr:text xmi:id="customer-t" value="customer"/>

<sbvr:term xmi:id="order" signifier="order-t" meaning="order-c"/>
<sbvr:objectType xmi:id="order-c"/>
<sbvr:text xmi:id="order-t" value="order"/>

<sbvr:sententialForm xmi:id="customer-places-order" expression="cpo-t" meaning="cpo-c" placeholder="cpo-p1 cpo-p2"/>
<sbvr:binaryFactType xmi:id="cpo-c" role="cpo-r1 cpo-r2"/>
<sbvr:designation xmi:id="places" signifier="places-t" meaning="cpo-c"/>
<sbvr:text xmi:id="cpo-t" customer places order"/>
<sbvr:text xmi:id="places-t" value="places"/>
<sbvr:placeholder xmi:id="cpo-p1" expression="customer-t"  meaning="cpo-r1"/>
<sbvr:placeholderUsesDesignation placeholder="cpo-p1" designation="customer"/>
<sbvr:factTypeRole xmi:id="cpo-r1"/>
<sbvr:placeholder xmi:id="cpo-p2" expression="order-t"  meaning="cpo-r2"/>
<sbvr:placeholderUsesDesignation placeholder="cpo-p2" designation="order"/>
<sbvr:factTypeRole xmi:id="cpo-r2"/>
```

Figure 5.21 : Exemple de vocabulaire généré dans RuleCNL

5.5 Outil support de RuleCNL

5.5.1 Implémentation

Une caractéristique importante d'un langage contrôlé fiable est son outil support. L'un des problèmes les plus importants (sinon le plus gros problème) des langages contrôlés est la facilité d'utilisation de ce nouveau langage par les utilisateurs finaux (Kuhn, 2010). En réalité, l'expressivité limitée en raison des restrictions imposées sur le vocabulaire et la syntaxe par les règles de la grammaire de ces langages contrôlés conduit à la difficulté d'écrire des phrases conformes à ces langages. L'écriture des phrases syntaxiquement correctes sans outils d'aide est beaucoup trop compliquée parce que l'utilisateur a besoin d'apprendre les restrictions sur la syntaxe, qui dans de la plupart des cas n'est pas trivial à expliquer. Ainsi, nous avons développé un outil support pour RuleCNL avec des éditeurs prédictifs qui rendent le processus d'écriture et la convivialité de RuleCNL aussi facile que possible.

Le prototype de l'outil support a été implémenté sur la plateforme open source Eclipse qui est un environnement de développement intégré (IDE). Nous l'avons étendu avec trois plug-ins supplémentaires : EMF[32] (*Eclipse Modeling Framework*) (EMF), Xtext et Xtend. La figure 5.21 présente l'architecture d'implémentation de RuleCNL.

EMF (Steinberg et al., 2008) est un framework de modélisation couplé avec des supports de génération de code pour la construction des outils et d'autres applications basées sur des modèles de données structurées. À partir d'une spécification de modèle décrite en XMI, EMF fournit des outils et un support d'exécution pour produire un ensemble de classes Java pour le modèle, avec un ensemble de classes d'adaptateurs (*adapter classes*) qui permettent la visualisation, l'édition et un éditeur de base. EMF est compatible avec les standards de modélisation de l'OMG que nous avons présenté au Chapitre 3 tels qu'UML, MOF, XMI. Il prend en charge le concept clé du MDA qui est d'utiliser des modèles en entrée pour les outils de développement et d'intégration. Dans EMF, un modèle est utilisé pour entraîner la génération de code et la sérialisation pour l'échange de données.

[32] EMF - http://www.eclipse.org/modeling/emf

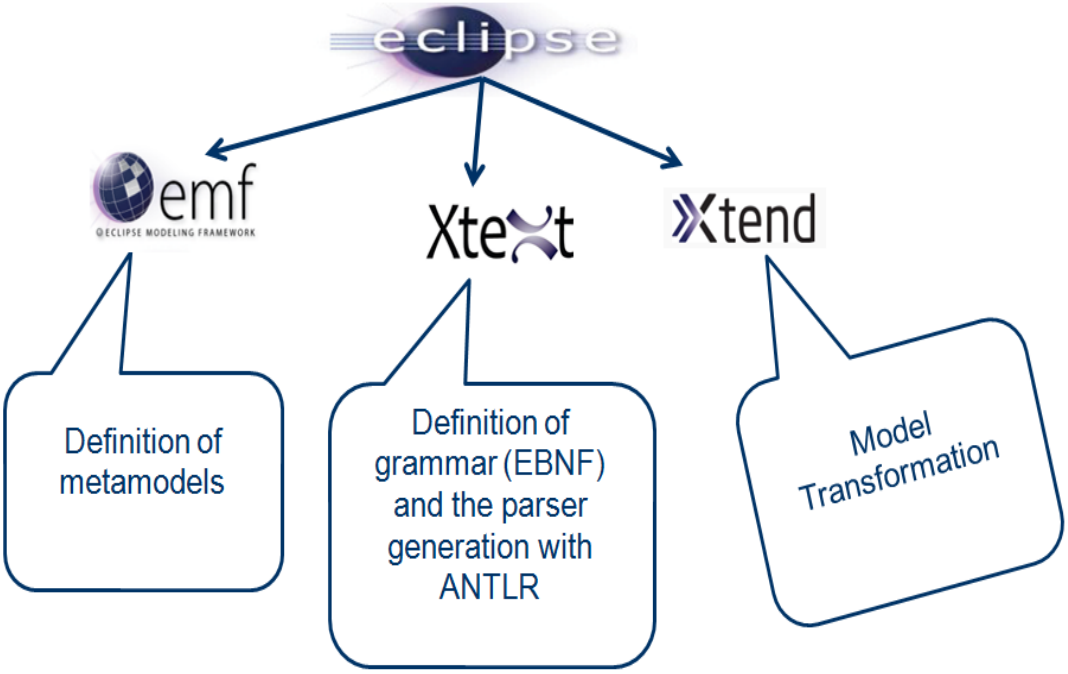


Figure 5.22 : Architecture d'implémentation de RuleCNL

Xtext[33] est un framework pour le développement de langages de programmation et des langages dédiés. À partir de la spécification d'une grammaire en notation EBNF, Xtext offre par défaut de bonnes infrastructures du langage comme un parseur et un éditeur basé sur Eclipse qui peuvent être adaptés à vos besoins.

Xtend[34] est un langage flexible et expressif basé sur le langage Java pouvant être utilisé pour les transformations de modèles et la génération de code. Il s'agit plus ou moins d'un langage fonctionnel pour l'interrogation et la navigation des modèles existants ainsi que la construction de nouveaux modèles. Le sous-langage d'expression est une version simplifiée d'OCL qui est un langage déclaratif de l'OMG préconisé pour les transformations de modèles dans l'approche MDA.

Ainsi, sur la base de ces frameworks, nous avons implémenté l'outil support de RuleCNL qui fournit aux experts métier des éditeurs conviviaux (vocabulaire et des règles éditeurs) avec les caractéristiques de hauts niveaux suivantes :

[33] Xtext - http://www.eclipse.org/Xtext

[34] Xtend - http://www.eclipse.org/xtend

(1) **Coloration syntaxique automatique.** Quand l'utilisateur est en train d'écrire une règle, l'éditeur met automatiquement en évidence les concepts du langage (*Domain Concepts*, *Domain Verbs* et *Rule Concepts*). Les styles de police de caractères par défaut sont ceux du *SBVR-Structured English*, mais l'utilisateur a également la possibilité de les changer et définir ceux de son choix. Cette caractéristique est d'une grande importance parce que dès qu'une entrée donnée ne peut être mise en évidence, l'utilisateur dispose d'une rétroaction instantanée qui lui indique qu'il y a soit une erreur dans la règle, ou que la fonction utilisée n'est pas prise en compte par notre langage RuleCNL.

(2) **Auto-complétion.** Dans la Section 5.3.3.1, nous avons vu que l'une des exigences fondamentales des langages contrôlés était la possibilité d'avoir des éditeurs prédictifs avec les caractéristiques de *look-ahead,* c'est-à-dire la possibilité de déduire les mots pouvant suivre un texte partiel. Ainsi, les éditeurs de l'outil support de notre RuleCNL offrent cette propriété de prédiction. À tout moment pendant le processus d'écriture d'un *Domain Term*, *Domain Name*, *Domain Verb* dans l'éditeur du vocabulaire ou d'une règle métier dans l'éditeur de règle, l'utilisateur peut appuyer sur *Ctrl + Espace* pour obtenir les options pour les prochaines entrées possibles comme le montre la figure 5.26.

(3) Vues personnalisées et dynamiques. L'éditeur du vocabulaire présente deux vues personnalisées et dynamiques sur le vocabulaire en cours de spécification qui sont mises à jour automatiquement. Une vue arborescente (cf. figure 5.23) et une vue graphique sous forme de diagramme de classe UML (cf. figure 5.24). L'utilisateur peut également accéder à un aperçu rapide dans l'éditeur en appuyant sur *Ctrl + O.*

(4) **Gestion d'erreurs et validation du modèle.** Pendant le processus d'écriture d'une règle, l'éditeur de règle vérifie instantanément et automatiquement la validité de la règle. À la fin de la spécification de la règle (marquée par un point), si elle n'est pas correcte alors des messages d'erreurs sont affichés comme le montre la figure 5.27. Une fois le modèle de règles métier de RuleCNL validé, le moteur de transformation est automatiquement lancé afin de générer les structures sémantiques SBVR du vocabulaire et des règles métier comme nous pouvons le voir respectivement sur les figures 5.28 et 5.29.

5.5.2 Étude de cas

Pour une étude de cas concrète, considérons les trois règles métier suivantes dans le domaine de la banque :

(1) **Simple :** *each customer may place at least one order.*

(2) **Complexe 1 :** *each customer must place an order if he is adult (age above 18).*

(3) **Complexe 2 :** *an order must be shipped if the customer who places is adult and the outstanding balance of its account is positive (above 0).*

Il s'agit ici de décrire le processus de formalisation de ces règles avec notre RuleCNL comme ce que ferait un expert métier (analyste banquier). La figure 5.22 présente le schéma de l'étude de cas.

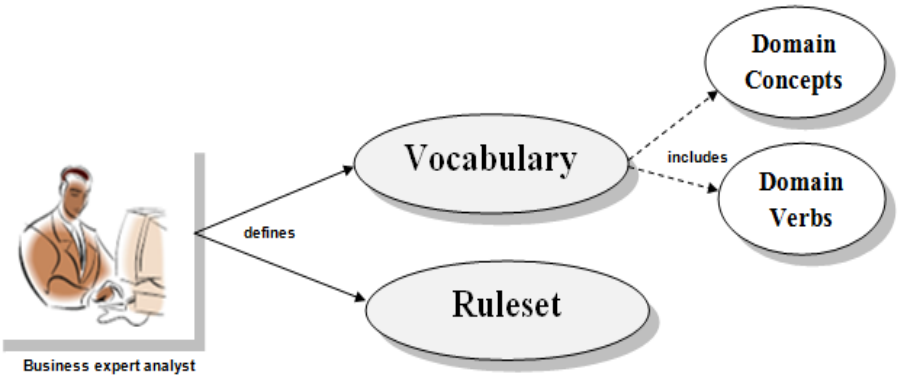


Figure 5.23 : Étude de cas principal de RuleCNL

Comme nous avons présenté plus haut, la première étape consiste à spécifier le vocabulaire c'est-à-dire les *Domain Concepts* et les *Domain Verbs.*

Les *Domain Concepts* et plus précisément les *Domain Terms* sont les suivants :

customer
order
account

Les *Domain Verbs* et plus précisément les *Domain Verbs* unaires sont les suivants :
order *shipped*
customer *adult*

Les *Domain Verbs* et plus précisément les *Domain Verbs* binaires sont les suivants :

customer *places* order
customer *holds* account
customer *has* age

account *has* outstanding balance

Nous avons aussi un *Domain Verb* binaire prédéfini à savoir :

quantity_1 *is greater than* quantity_2

La deuxième étape consiste à spécifier maintenant les règles métier elles mêmes basées sur le vocabulaire décrit précédemment.

(1) **Simple : It is necessary that each** customer *place* **at least 1** order.

(2) **Complexe 1 :**

It is obligatory that each customer *place* **an** order **if the** age *of* **the** customer *is greater than* **0.**

(3) **Complexe 2 :**

It is obligatory that each order *be shipped* if the customer **who** *places* **the** order *is adult* **and** *holds* **an** account **that** *has* **an** outstanding balance **that** *is greater than* **0**.

Nous pouvons voir dans la spécification des règles ci-dessus que les *Domain Verbs* sont explicitement et soigneusement bien présents dans les règles afin d'éviter toute ambigüité. Ces exemples ont été effectivement implémentés et nous pouvons voir sur les figures 5.23 et 5.24 les éditeurs de vocabulaire ; sur les figures 5.25, 5.26 et 5.27 les éditeurs de règles métier et sur les figures 5.28 et 5.29 les éditeurs de vocabulaire et de règles métier générés en modèles sémantiques SBVR.

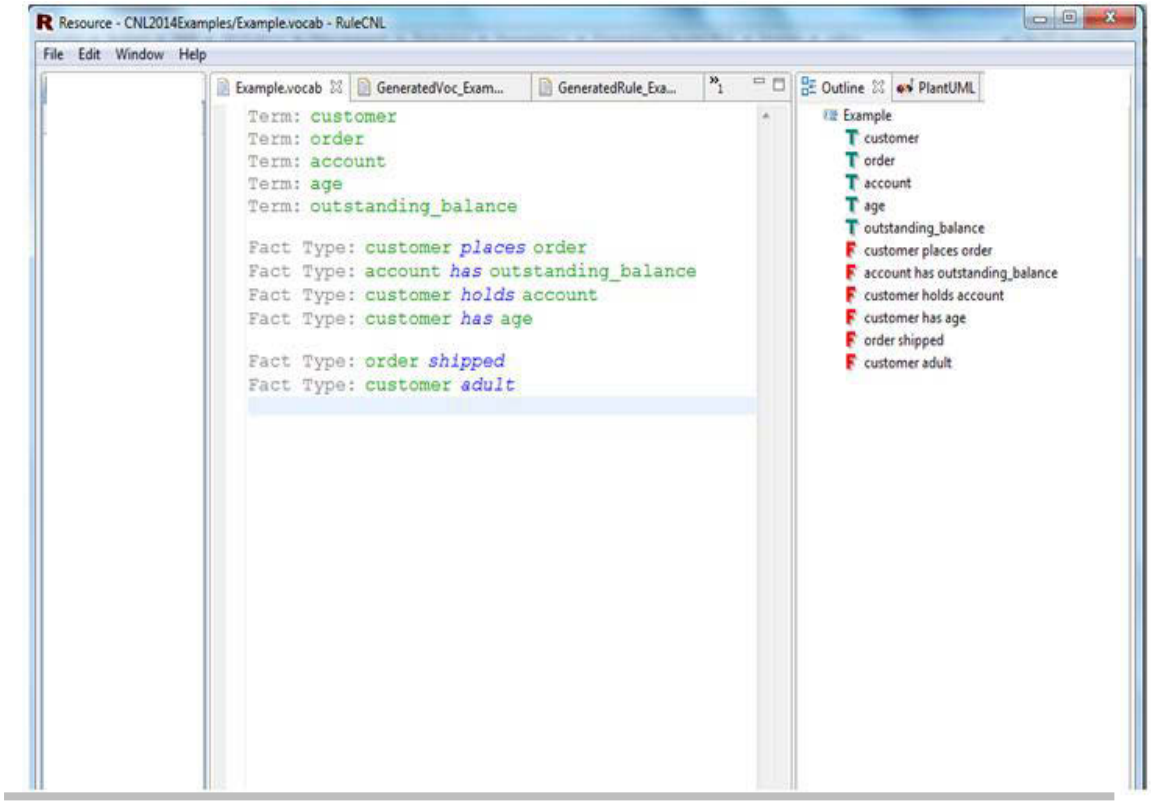


Figure 5.24 : Éditeur de vocabulaire de RuleCNL avec une vue arborescente

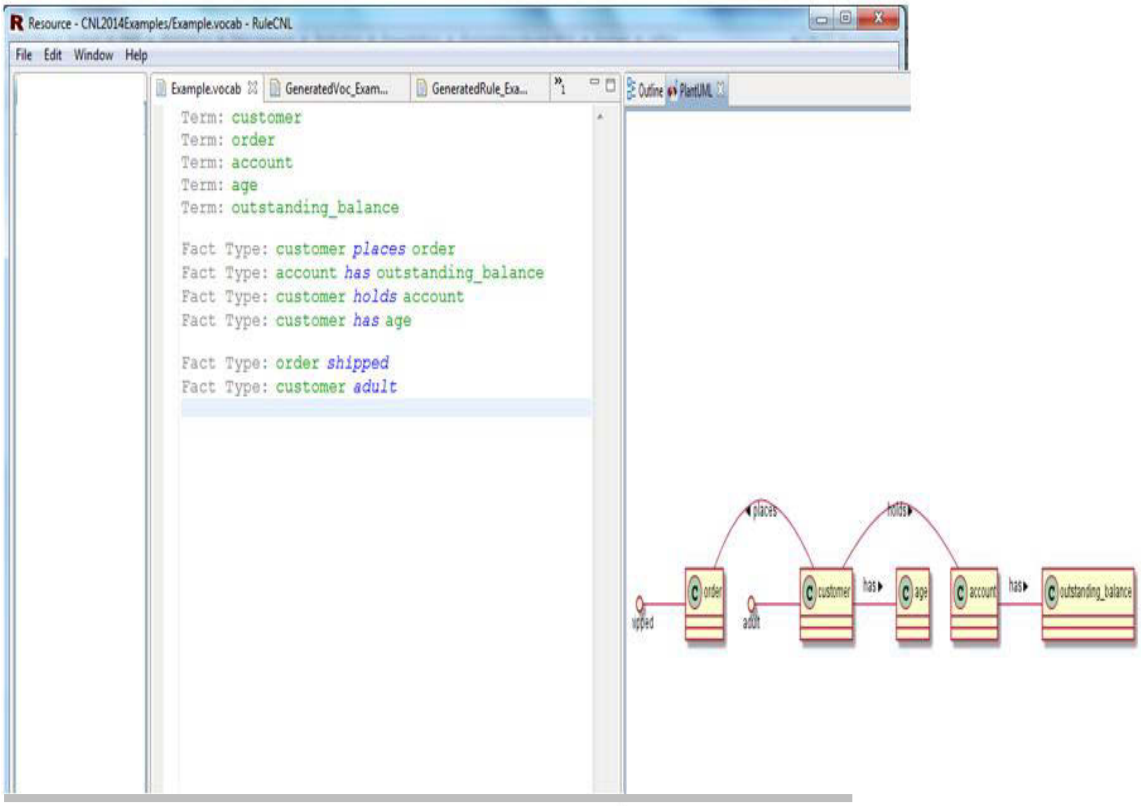


Figure 5.25 : Éditeur de vocabulaire de RuleCNL avec une vue graphique

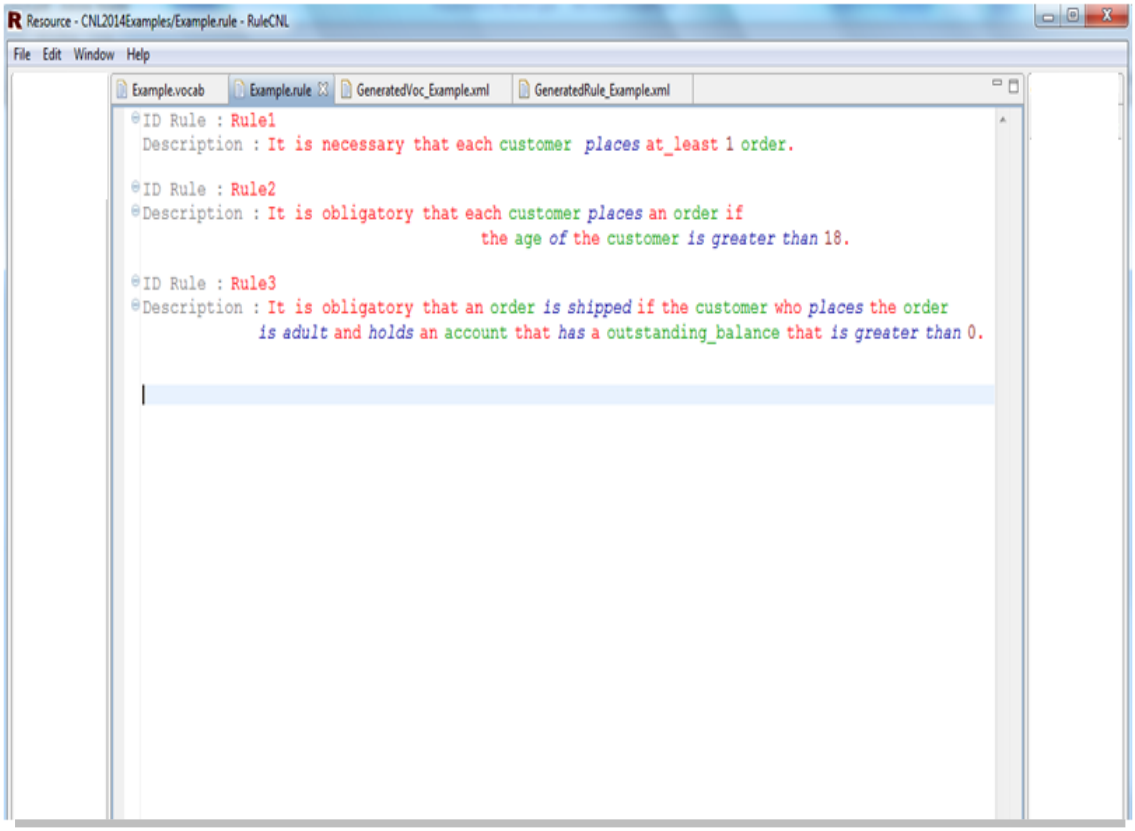


Figure 5.26 : Éditeur de règles métier de RuleCNL

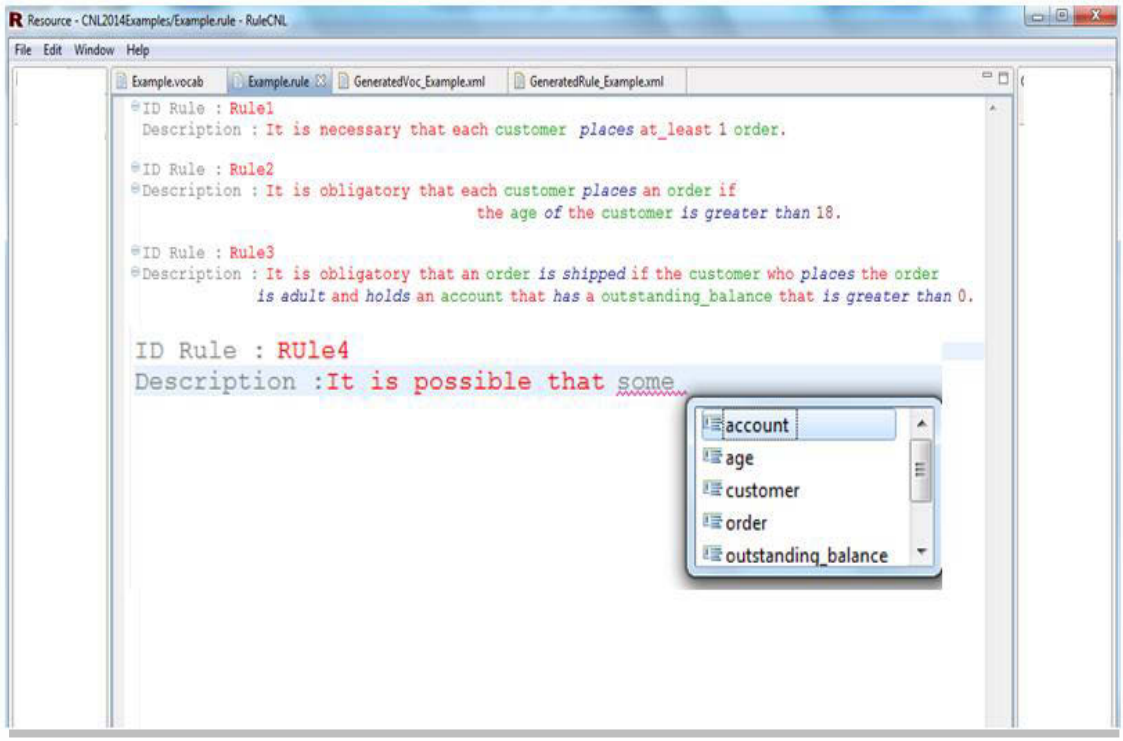


Figure 5.27 : Éditeur de règles métier de RuleCNL avec auto-complétion

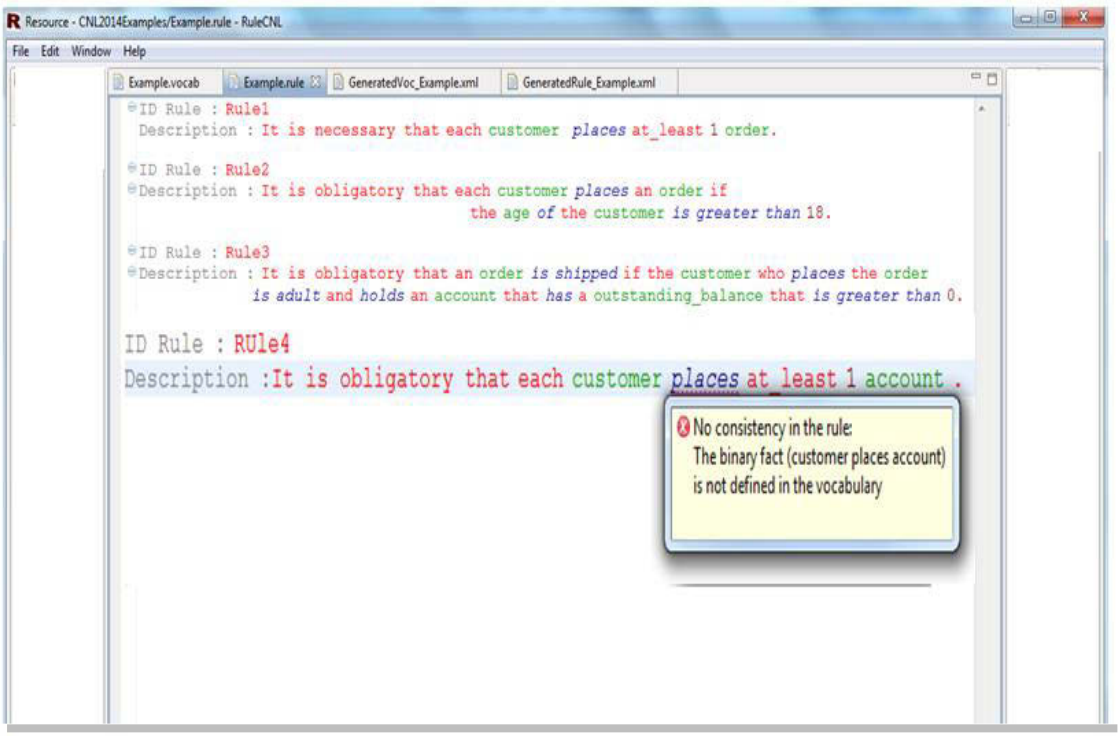


Figure 5.28 : Éditeur de règles métier de RuleCNL avec gestion d‟erreurs

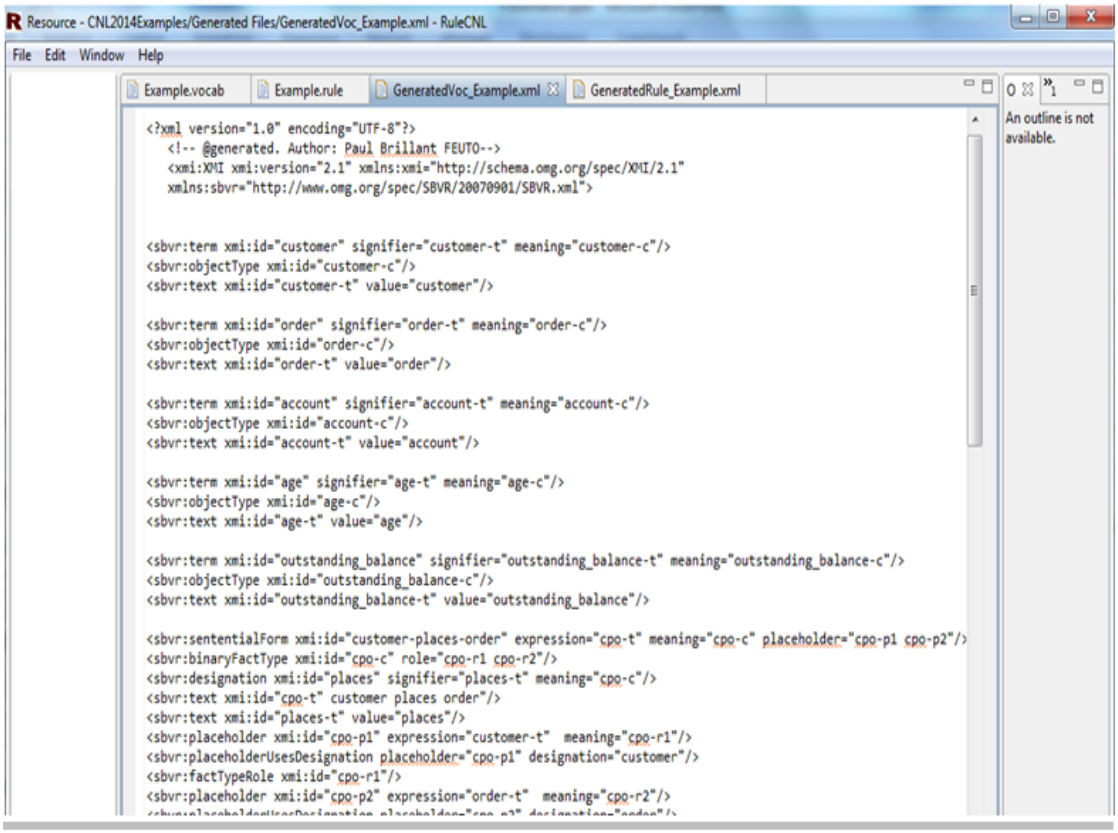


Figure 5.29 : Vocabulaire généré dans RuleCNL

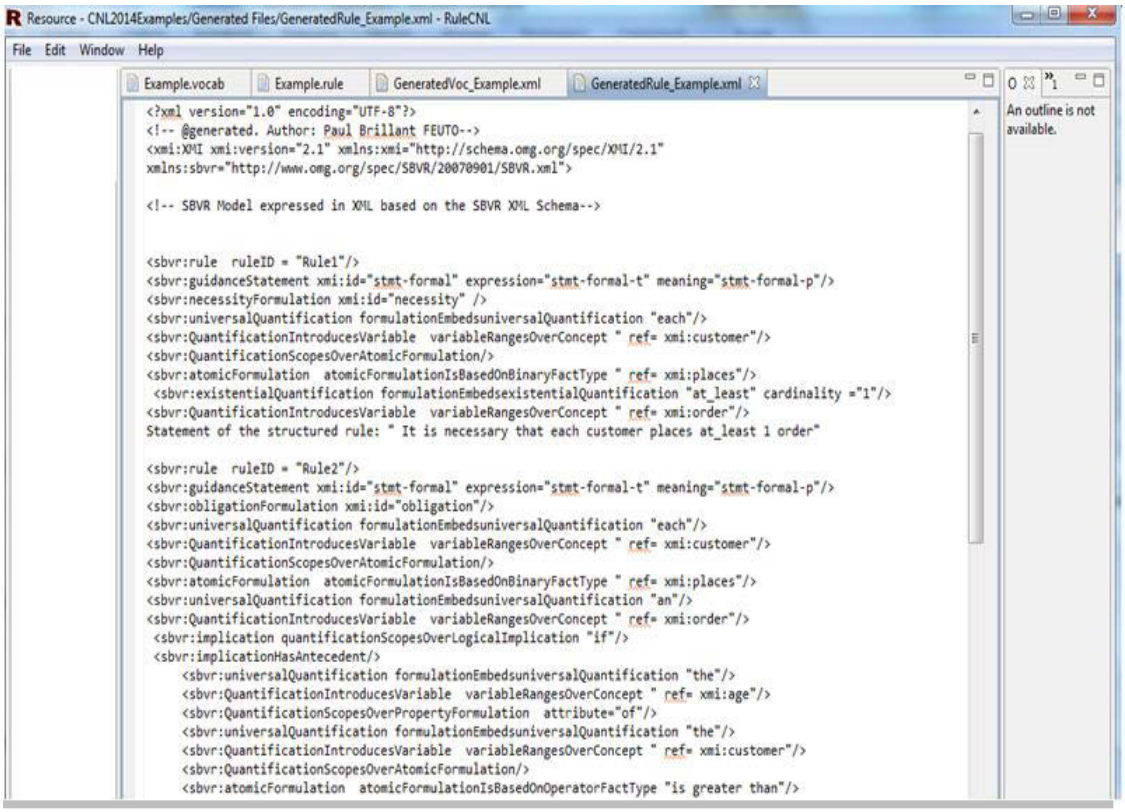


Figure 5.30 : Règles métier générées dans RuleCNL

5.5.3 Évaluation de RuleCNL

Un langage contrôlé idéal devrait être facile à apprendre et suffisamment expressif pour décrire le problème du domaine. Ainsi, nous avons évalué notre langage contrôlé RuleCNL par rapport à trois critères fondamentaux : son expressivité, son intelligibilité et sa lisibilité. Pour la mise en œuvre de l'évaluation, nous avons constitué un corpus d'environ 50 règles métier à partir d'une étude de cas réelle de deux sociétés. La première société opère dans le domaine de la banque et de l'assurance tandis que la seconde est un organisme parapublic. Les règles du corpus sont écrites en français et en anglais. Ce corpus était vraiment significatif pour notre expérimentation car il possède une large couverture en termes de la typologie et de la structure syntaxique des règles. L'évaluation a été réalisée par des experts métier répartis en deux groupes. Le groupe 1 est composé des experts métier sans aucune connaissance dans les notations formelles de règles métier et le groupe 2 est constitué des experts métier avec un profil en technologie des systèmes d'informations.

Notre objectif était double et consistait à trouver combien de règles métier notre RuleCNL pourrait formaliser d'une manière naturelle et à montrer la facilité avec laquelle ces utilisateurs pourraient comprendre la formalisation. Ainsi, nos paramètres d'évaluation sont l'expressivité, la compréhension et la lisibilité du langage RuleCNL.

Le tableau 5.3 montre le résultat de l'expérience avec l'accord des utilisateurs de chaque groupe. Ce résultat est le même à la fois pour l'anglais et le français.

Measures / ***Users***	***Group 1***	***Group 2***
Expressiveness	84%	84%
Comprehensibilty	90%	100%
Readability	100%	100%

Tableau 5.3 : Évaluation de RuleCNL

Comme nous pouvons le voir dans le tableau 5.3, l'expressivité est de 84% pour les deux groupes. Le pourcentage de 16% restant est dû aux ambiguïtés syntaxiques et sémantiques liées à certaines règles. Cependant, avec plus de formation et d'explication sur les règles de syntaxe de RuleCNL, les utilisateurs ont pu reformuler ces règles afin que l'outil de RuleCNL soit en mesure de les formaliser. En ce qui concerne la compréhension, le groupe 1 a confirmé qu'il comprenait 90% des règles formalisées de façon naturelle et le groupe 2 comprenait toutes les règles. Ce résultat n'est pas surprenant, car les utilisateurs du groupe 1 n'ont pas de connaissances en notations formelles. Ainsi, le pourcentage de 10% de règles restantes était composé des règles complexes, qui nécessitaient beaucoup de contraintes imposées par la grammaire de RuleCNL. Pour ce qui est de la lisibilité, tous les deux groupes ont été capables de bien lire les règles de façon naturelle. En effet, comme RuleCNL est un sous-ensemble

propre du langage naturel, les expressions de règles écrites dans RuleCNL sont lues de la même manière que dans le langage naturel sous-jacent.

5.6 Conclusion

Dans ce chapitre, nous avons présenté la principale contribution de cette thèse qui consiste à proposer le langage contrôlé que nous avons appelé RuleCNL pour la spécification des règles métier. Les règles métier jouent un rôle fondamental dans les entreprises qui se veulent être modernes c'est-à-dire compétitives et agiles face aux changements. Elles représentent le moyen principal de capturer la logique métier gérée par les experts métier et de rendre les systèmes d‟informations flexibles. Un langage contrôlé naturel (bon médiateur entre le langage naturel des experts métier et les langages formels des règles) pour la spécification de ces règles métier permet notoirement un meilleur alignement de la logique métier et de la logique système. Dans ce contexte, nous avons présenté notre RuleCNL qui contribue dans cette direction. Nous avons à tour de rôle présenté sa syntaxe (abstraite et concrète), sa sémantique d‟exécution, son outil support, une étude de cas ainsi que son évaluation. Les résultats de l‟évaluation présentent des résultats satisfaisants et montrent bien que RuleCNL pourrait trouver un écho favorable auprès de plusieurs entreprises.

Chapitre 6

Conclusions et Perspectives

6.1 Conclusions

Cette section conclut les recherches menées afin de démontrer notre hypothèse de départ. Trois axes de recherche nous ont été nécessaires en lien avec la modélisation des systèmes d'information à savoir : l'approche par règles métier, l'ingénierie dirigée par les modèles et les langages contrôlés. L'hypothèse de départ posée à la Section 1.4 était :

Comment réduire la distance linguistique et sémantique entre le langage naturel des experts métier et le langage formel des règles métier en passant par la sémantique SBVR ?

Nos contributions à cette thèse passent par la mise en œuvre d'un langage contrôlé que nous avons appelé RuleCNL pour la spécification des règles métier ainsi qu'un moteur de transformations automatiques vers des modèles sémantiques SBVR. Ainsi, les règles métier sont exprimées à la fois de façon naturelle compréhensible par les experts métier et les experts systèmes, et de façon formelle interprétable par des machines.
Avant de rappeler le détail de nos contributions, il est important de souligner ici que notre hypothèse découle d'une problématique que nous avons évoquée à la Section 1.2 du chapitre introductif et qui a suscité des recherches que l'ont trouvent dans la littérature. Cette problématique était la suivante :

Dans le contexte de la modélisation des SI, comment séparer la logique métier basée sur les règles métier et la logique système à savoir les applications métier ?

Cette question concerne les limitations des approches traditionnelles de modélisation des SI et souligne la nécessité de voir comment externaliser les règles métier des processus et des applications métier automatisés afin de les gérer de façon centralisée par les experts métier. Elles pourront ainsi être gérées par les experts métier comme des citoyens de première classe

dans l'entreprise et partagées de façon formelle dans différentes applications métier dans un souci de consistance, de traçabilité et de cohérence. Aussi, les règles métier pourront évoluer à la vitesse des contraintes de l'environnement métier, et non à la vitesse des contraintes des plateformes technologiques. La réponse à cette problématique a fait l'objet des chapitres 2 et 3 dans lesquels nous avons présenté respectivement l'approche par règles métier et l'ingénierie dirigée par les modèles autour du *Model Driven Architecture*. Ce sont deux approches éprouvées et complémentaires qui ont contribué dans cette direction. Dans la modélisation des SI, l'approche par règles métier préconise la séparation de la gestion des règles métier des applications métier. Le *Model Driven Architecture* propose une architecture qui promeut la modélisation des modèles/langages de règles métier à trois niveaux d'abstractions (CIM, PIM et PSM). Le *Model Driven Architecture* propose aussi deux principes fondamentaux que sont la méta-modélisation et les transformations de modèles. La méta-modélisation est l'activité permettant de définir des langages de modélisation appropriés à ces différents niveaux d'abstractions. Les transformations de modèles permettent de passer des modèles d'un niveau d'abstraction à un autre. Le principal lien entre l'approche par règles métier et le *Model Driven Architecture* est le méta-modèle SBVR qui est le standard clé de l'ARM et appartient à la couche des modèles métier CIM du *Model Driven Architecture*

Cependant, au delà de ces deux approches séduisantes, l'enjeu majeur reste la gestion de l'ambigüité du langage métier pour permettre l'expression formelle des règles métier par les experts métier au niveau de la couche CIM et leurs transformations vers des modèles PIM et ensuite vers des modèles PSM pour les moteurs de règles dans une approche par règles métier. Il existe toujours une distance sémantique entre les modèles/langages métier CIM et les modèles/langages PIM qui est d'ailleurs une limitation héritée de l'approche MDA car elle n'émet aucune préconisation quant à l'élaboration des modèles métier CIM et leurs transformations vers des modèles PIM (cf. figure 1.2). Ainsi, la réduction de cette distance sémantique entre les modèles/langages métier et les modèles/langages PIM utilisés dans le processus de développement des SI constitue un enjeu fondamental pour une méthodologie efficace et complète. Ceci nous conduit à notre hypothèse que nous pouvons reformuler ainsi comme suit :

Comment restreindre le langage naturel (langage contrôlé) des experts métier afin de leur permettre de pouvoir eux-mêmes modéliser leur modèles métier CIM (vocabulaire et règles

métier) de façon formelle pour ensuite les transformer vers des modèles PIM, puis vers des modèles concrets (PSM) en vue de leur implémentation dans les SI ?

À partir de là et suivant un processus MDA (cf. figure 1.2), les experts système ne s'occuperont que de la logique applicative et des aspects liés à leur déploiement sur des plateformes d'exécutions. Ceci permettra de réduire la distance linguistique et sémantique entre les experts métier et les experts système afin d'éviter des défaillances et des incohérences dans les SI liées aux ambigüités de communication en langage naturel.

Cette hypothèse de thèse souligne le problème général de la modélisation en langage naturel qui est un problème récurent dans le génie logiciel dû aux ambigüités du langage naturel. Nous nous sommes tournés vers les langages contrôlés qui ont pour but ultime de réduire ou contourner les ambigüités du langage naturel et aussi d'interfacer les langages formels existants. C'est ainsi que nos contributions consistent à proposer un langage contrôlé que nous avons appelé RuleCNL pour la spécification des règles métier ainsi qu'un moteur de transformations automatiques vers des modèles sémantiques SBVR. À partir des modèles sémantiques SBVR, on pourra aller vers n'importe quels autres modèles PIM et PSM.

Au Chapitre 4, nous avons tout d'abord fait un tour d'horizon sur les recherches dans le domaine des langages contrôlés en général et des langages contrôlés pour les règles métier en particulier. Comme nous avons décrit à la Section 4.4 et illustré à la figure 4.2, il n'existe pas à notre connaissance assez de langage contrôlé naturel pour l'expression des règles métier à destination des experts métier (cf. figure 4.2, couche CIM). *SBVR Structured-English* et *RuleSpeak* qui sont les plus cités dans la littérature ne sont pas des langages en soi, mais plutôt des ensembles de bonnes pratiques dans les systèmes de règles pour les locuteurs humains (Kuhn, 2010). Ils sont définis par ses auteurs comme des ensembles de directives pour exprimer des règles métier de façon claire et concise. Ils sont définis de façon informelle et n'ont pas de syntaxe (grammaire formelle) et par conséquent ne peuvent être traités de manière entièrement automatique (Ruth, 2009). Ainsi, des efforts supplémentaires doivent être fournis afin de les formaliser (définir une grammaire formelle) pour un traitement automatique.

C'est sur cette base que nous avons proposé au Chapitre 5 RuleCNL pour la spécification des règles métier. Notre méthodologie est basée sur l'alignement de la définition des règles avec le vocabulaire du métier. Notre langage contrôlé RuleCNL est conçu suivant une approche de

méta-modélisation et de transformations des modèles basée sur MDA. Il est défini par une syntaxe formelle et par une sémantique d'exécution. Les détails de nos contributions sont les suivants :

- la catégorisation des règles métier du point de vue de leurs structures syntaxiques ;

- la définition des méta-modèles des règles et du vocabulaire métier en vue de la spécification de la syntaxe abstraite du langage ;

- la définition des contraintes (sémantique statique) pour l'alignement de la définition des règles métier avec le vocabulaire du métier ;

- la définition d'une grammaire formelle hors-contexte en vue de la spécification de la syntaxe concrète du langage ;

- la définition des transformations automatiques des modèles de règles de RuleCNL vers des formulations sémantiques SBVR qui constitue la sémantique d'exécution. Ceci est le résultat d'un long effort de compréhension de la spécification SBVR ;

- l'implémentation de l'outil support de RuleCNL afin d'aider les experts métiers à formaliser leurs règles métier d'une manière conviviale et qui peuvent être interprétées par les ordinateurs. L'outil support de RuleCNL fournit des éditeurs avec des interfaces conviviales qui assistent les experts métier dans le processus d'écriture ;

- l'évaluation de RuleCNL ainsi que son outil support auprès des experts métier.

6.2 Perspectives

Comme perspectives de nos travaux, nous aimerions dans un futur proche :

- étendre l'évaluation de notre langage contrôlé RuleCNL et de son outil support auprès de plusieurs entreprises et organismes afin d'améliorer son efficacité ;

- étendre aussi le méta-modèle de RuleCNL ainsi que sa grammaire en cherchant à définir cette dernière avec certains frameworks plus appropriés pour le langage naturel que les grammaires hors-contexte comme le *Grammatical Framework*[35] ;

- améliorer l'outil support de RuleCNL et principalement ses éditeurs et y ajouter plus de fonctionnalités pour la gestion des règles telles que les droits d'accès, la gestion des versions, etc. ;

- à partir des modèles sémantiques SBVR générés par le moteur de transformations de l'outil support de RuleCNL, aller vers plusieurs applications en utilisant toujours le principe de la méta-modélisation et des transformations de modèles. Une application concrète sera de générer du code SQL à partir de règles métier afin de valider les transactions métier dans le domaine de la gouvernance des données. Ainsi, les SI seront agiles et alignés avec l'évolution des besoins de l'entreprise et ceci permettra de combler le pont sémantique entre l'entreprise et l'informatique.

[35] GF - http://www.grammaticalframework.org

Bibliographie

Abelson, H., Sussman, G. J., Sussman, J. (1996). *Structure and interpretation of computer programs*. 2nd edition, Cambridge, MA, USA: MIT Press, 657 p.

Afreen, H. et Bajwa, I. S. (2011). Generating UML Class Models from SBVR Software Requirements Specifications. In *23rd Benelux Conference on Artificial Intelligence (BNAIC 2011)*. Gent, Belgium, p. 23-32.

Angelov, K. et Ranta, A. (2009). Implementing Controlled Languages in GF. In *Proceedings of Controlled Natural Language (CNL 2009).* Springer Berlin Heidelberg, p. 82-101.

Association Européenne des Constructeurs de Matériel Aérospatial (1986). *AECMA Simplified English*. PSC-8S-16598.

Atkinson, C. et Kühne, T. (2002). The role of meta-modeling in MDA. *Workshop in Software Model Engineering (WISME@UML)*. Dresden, Germany.

Bajec, M. et Krisper, M. (2005). A methodology and tool support for managing business rules in organisations. *Information Systems*, 30, p. 423–443.

Bajwa, I. S., Bordbar, B., Lee, M. G. (2010). OCL Constraints Generation from Natural Language Specification. In *14th IEEE International Enterprise Distributed Object computing Conference*, Victoria, Brazil, p. 204-213.

Barbier, F. (2005). *UML 2 et MDE : Ingénierie des modèles avec études de cas*. Dunod, 361p.

Barthe, K. (1998). GIFAS Rationalised French: Designing One Controlled Language to Match Another. In *Proceedings of the Second International Workshop on Controlled Language Applications (CLAW98)*. Pittsburgh, Pennsylvania: Language Technologies Institute, Carnegie Mellon University, p. 87-102.

Barthe, K. (1996). EUROCASTLE – A User‟s Experience with Prototype AECMA SE Checkers. In *Proceedings of the First International Workshop on Controlled Language*

Applications CLAW96. Leuven, Belgium: Katholieke Universiteit Leuven Centre for Computational Linguistics, p. 42-63.

Bernardi, R., Calvanese, D., Thorne, C. (2007). Lite natural language. In *Proceedings of IWCS-7*. http://www.inf.unibz.it/~cathorne/perso/lite.pdf

Bernth, A. (1997). Easy English: a tool for improving document quality. In *Proceedings of the Fifth Conference on Applied Natural Language Processing*, Association for Computational Linguistics, p. 159-165.

Blanc, X. (2005). *MDA en action ingénierie logicielle guidée par les modèles*, Eyrolles, ISBN 2212115393, 260 p.

Blanc, X., Ramalho, F., Robin, J. (2005). Metamodel Reuse with MOF. In *MoDELS*, LNCS 3713. Springer Berlin Heidelberg, p. 661-675.

Blanco X. (2009). Remarks about Linguistic Analysis, Normalization and Translation of Spanish "What to Do in Case of Fire" Texts. In *ISMTCL Proceedings, International Review Bulag*, *PUFC,* p. 43-48.

Bogacki, K. (2009). Controlled Languages and Machine Translation. In *ISMTCL Proceedings, International Review Bulag*, *PUFC*, p. 49-55.

Boyer, J. et Mili, H. (2011). *Agile Business Rule Development: process, Architecture, and JRules Examples*. Springer, 567p.

Buchanan, B., Sutherland, G., Feigenbaum E. A. (1969). Heuristic Dendral: a program for generating explanatory hypotheses in organic chemistry. In *Proceedings of the Fourth Annual Machine Intelligence Workshop*. Edinburgh University Press, p. 209-254.

Budinsky, F., Steinberg, D., Ellersick, R. (2003). *Eclipse Modeling Framework: A Developer's Guide*. Addison-Wesley Professional.

Bunke, H., Glauser, T., Tran, T. H. (1990). An Efficient Implementation of Graph Grammars Based on the RETE Matching Algorithm. In *Graph-Grammars and Their Application to Computer Science*, p. 174-189.

Business Rule Group (2000). *Defining Business Rules ~What Are They Really?* Final Report, revision 1.3.

Business Rule Group, (2003). *Business Rule Manifesto*. V2.0, Ross, G. R. eds, http://www.businessrulesgroup.org/brmanifesto/BRManifesto.pdf.

Butleris, R. et Kapocius K. (2002). The business rules repository for information systems design. In *ADBI Research Communications of 6th East European Conference*. p. 64–77.

Cardey, S. et Greenfield, P. (2005). A core model of systemic linguistic analysis. In *Proceedings of the International Conference RANLP-2005 Recent Advances in Natural Language Processing*. Borovets, Bulgaria, 21-23 September, p. 134-138.

Cardey, S. et Greenfield, P. (2003). Disambiguating and tagging using systemic grammar. In *Actes du 8th International Symposium on Social Communication*, Santiago de Cuba, January 20-24, p. 559-564.

Cardey, S., Greenfield, P., Vienney, S. (2005). Machine translation, controlled languages and specialised Languages. In *lingvisticae Investigationes*, Benjamin.

Cardey, S. (2005). Introduction to machine translation, controlled languages and specialised languages, *In proceedings of the workshop on machine translation, controlled languages and specialised languages*, Besançon, 5-6 May. Lingvisticae Investigationes, Benjamins, p. 2-3.

Cardey, S. et Greenfield, P. (2006). Systemic linguistics with applications. In *Linguistics in the Twenty First Century*. United Kingdom : Cambridge Scholars Press, p. 261-271.

Cardey, S., Greenfield, P., Anantalapochai, R. et al. (2008). Modelling of multiple target machine translation of controlled languages based on language norms and divergences," In *Proceedings of ISUC2008 (Second International Symposium on Universal Communication)*, Osaka, Japan, December 15-16. IEEE Computer Society, p. 322-329.

Cardey, S. et Greenfield, P. (2008). Micro-systemic linguistic analysis and software engineering: a synthesis, In *revue RML6*, *Actes du Colloque International en Traductologie et TAL*, 7 et 8 juin 2008, Editions Dar El Gharb, Oran, ISBN: 978 9961 54 593 1, p. 5-25.

Cardey, S., Wu, X., Vuitton, D. et al. (2009). Controlled language: a Linguistic Concept to Improve Health Care Safety in a "Globalised" World? Application to Medical Protocols Written within the Hospital Accreditation/Certification Framework in France and China. In *ISMTCL Proceedings, International Review Bulag*. *PUFC*, p. 260-268.

Cardey, S., Bogacki, K., Blanco, X. et al. (2010). Resources for controlled languages for alert messages and protocols, in the European perspective. In *proceedings of LREC 2010*.

Cardey, S. (2011). Machine translation of controlled languages for more reliable human communication in safety critical applications. In *Proceedings of the 12th International Symposium on Social Communication* - Comunicación Social en el Siglo XXI, Santiago de Cuba, Cuba, January 17-21, p. 953-958.

Cardey, S. (2013). *Modelling Language (Natural Language Processing 10)*. John Benjamins Publishing Company. ISBN 9789027249968.

Chapin, D. (2005). Semantics of Business Vocabulary & Business Rules (SBVR). In *Rule Languages for Interoperability*.

Charles, A. et Verbeke, C. A. (1973). Caterpillar fundamental English. Training & Development Journal, 27(2): p. 36-40.

Cholewa, J. (2009). Remarks Concerning Writing Texts in Controlled Polish. In *ISMTCL Proceedings, International Review Bulag, PUFC*, p. 62-68.

Clark, P., Murray, W. R., Harrison, P. et al. (2010). Naturalness vs. predictability: A key debate in controlled languages. In *Controlled Natural Language*. Springer Berlin Heidelberg, p. 65-81.

Combemale, B. (2008). *Approche de métamodélisation pour la simulation et la vérification de modèle : Application à l'ingénierie des procédés*. Thèse de doctorat en Informatique, Université de Toulouse, Toulouse.

Crabbe, S. (2009). Controlled languages for technical writing and translation. In *Proceedings of the 9th Portsmouth Translation Conference*, p. 48–62.

Cramer, M., Fisseni, B., Koepke, P. et al. (2010). The naproche project controlled natural language proof checking of mathematical texts. In *Controlled Natural Language*. Springer Berlin Heidelberg, p. 170-186.

Cregan, A. Schwitter, R., Meyer, T. (2007). Sydney OWL Syntax – towards a controlled natural language syntax for OWL 1.1. In *Proceedings of the OWLED 2007 Workshop on OWL: Experiences and Directions*. Vol. 258 of CEUR Workshop Proceedings.

Czarnecki, K. et Simon Helsen, S. (2003). Classification of Model Transformation Approaches. In *OOPSLA'03 Workshop on Generative Techniques in the Context of Model-Driven Architecture*.

Date, C. J. (2000). *What Not How: The Business Rules Approach to Application Development*, Reading MA: Addison-Wesley.

Demuth, B., Liebau, H. B. (2007). An Approach for Bridging the Gaps Between Business Rules and the Semantic Web. In *RuleML 2007. LNCS*, vol. 4824. Springer Berlin Heidelberg, p. 119–133.

Denaux, R. (2013). *Intuitive Ontology Authoring using Controlled Natural Language*. Doctoral thesis, School of Computing, University of Leeds.

Diouf, M. Musumbu, K., Maabout, S. (2006). Standard Business Rules Language: why and how? In *the 2006 International Conference on Artificial Intelligence*.

El Abed, W. (2000). *Méta modèle sémantique et noyau informatique pour l'interrogation multilingue des bases de données en langue naturelle (théorie et application)*. Thèse de doctorat, Centre de Recherche Lucien Tesnière, Université de Franche-Comté, Besançon.

Feuto Njonko, P. B. (2011). Rule based approach for normalizing messages in the security domain. In *Natural Language Processing and Human Language Technology 2011, BULAG* n°36, PUFC, ISSN 0758 6787.

Fondement, F. (2007). *Concrete syntax definition for modeling languages*. Thèse de doctorat, école polytechnique fédérale de Lausanne, Lausanne.

Forgy, C. (1982). Rete: A Fast Algorithm for the Many Patterns/Many Objects Match Problem. *Artif. Intell*., 19 (1) : p. 17-37.

Fuchs, N.E. et Schwitter, R. (1995). Specifying logic programs in controlled natural language. In *Proceedings of CLNLP 95*.

Fuchs, N. E., Kaljurand, K., Kuhn, T. (2008). Attempto Controlled English for knowledge representation. In *Reasoning Web*, Springer Berlin Heidelberg, p. 104–124.

Funk, A., Tablan, V., Bontcheva, K. et al. (2007). Clone: Controlled language for ontology editing. *In Proceedings of the ISWC 2007 + ASWC 2007*, vol. 4825. Springer Berlin Heidelberg, p. 142-155.

Frankel, D. S. (2003). *Model Driven Architecture: Applying MDA to Enterprise Computing*. Wiley, ISBN: 978-0-471-31920-7, 328 p.

Friedman-Hill, E. (2003). *JESS in Action*. Manning Publications Co, Greenwich, UK.

Fu, G., Shao, J., Embury, S.M. et al. (2004). Algorithms for analyzing related constraint business rules. *Data & Knowledge Engineering*, 50, p. 215–240.

Gawel, B. et Skalna, I. (2012). Model Driven Architecture and classification of business rules modelling languages. In *Proceedings of the Federated Conference on Computer Science and Information Systems (FedCSIS).* IEEE, p. 949-952.

Gottesdiener, E. (1997). Business RULES show power, Promise. In *Issue of Application Development Trends,* vol. 4., no. 3.

Grosof, B. N. et Labrou, Y. (2000). *An approach to using XML and a rule-based content language with an agent communication language*. Springer Berlin Heidelberg, p. 96-117.

Gupta, C., Forgy, A., Newell, A. (1989). High-speed implementations of rule-based systems. *ACM Transactions on Computer Systems (TOCS)*, 7(2), p. 119-146.

Gwiazdecka E., (2009). Annotation of Terminology from Protocols in Polish Controlled Language. In *ISMTCL Proceedings, International Review Bulag, PUFC*, ISBN 978-2- 84867-261-8, p. 121-126.

Hadj Kacem, M. (2008). *Modélisation des applications distribuées à architecture dynamique : Conception et Validation*. Thèse de doctorat en Informatique, Université de Toulouse et Université de Sfax.

Halpin, T. A. (2004). *Business rule verbalization*. In *Proceedings of the 3rd International Conference ISTA 2004*, volume P-48 of Lecture Notes in Informatics, p. 39-52.

Hamza, H.S. et Fayad, M. E. (2005). A novel approach for managing and reusing business rules in business architectures. In: Hamza, H.S. & Fayad, M.E. (Eds.). AICCSA 2005 Workshop: In *Association with the 3rd ACS/IEEE International Conference on Computer Systems and Applications*, AICCSA05. Cairo, Egypt.

Harel, D. et Rumpe, B. (2004). Meaningful Modeling: What‟s the Semantics of "Semantics"?. *Computer 37* no. 10, p. 64–72.

Harold, E.R. (2001). *The XML Bible*. Hungry Minds Inc. New York, USA.

Hart, G., Johnson, M., Dolbear, C. (2008). Rabbit: Developing a control natural language for authoring ontologies. In *Proceedings of ESWC 2008*, vol. 5021. Springer Berlin Heidelberg, p. 348–360.

Hay, D. et Healy, K. A. (2000). *Defining Business rules What Are They Really*? Technical Report 1.3, The Business Rules Group.

Hendrix, G. G., Sacerdoti, E. D., Sagalowicz, D. et al. (1978). Developing a natural language interface to complex data. *ACM Transactions on Database Systems (TODS)*, 3(2), p. 105-147.

Herbst, H. (1996). Business rules in systems analysis: A meta-model and repository system. *Information Systems*, 21, p. 147–166.

Herbst, H., Knolmayer, G., Myrach, T. et al. (1994). The Specification of Business Rules: A Comparison of Selected Methodologies. In *Methods and Associated Tools for the Information Systems Life Cycle*, p. 29-46.

Hoffman, T. (1999). Study: 85% of IT departments fail to meet biz needs. *Computer World.*

Horridge, M., Drummond, N., Goodwin, J. et al. (2006). The Manchester OWL Syntax. In *Proceedings of OWLed*, vol. 216.

Huber, V. (1997). Automating Business Rules (interview), Database Newsletter, 25, No. 2.

Huijsen, W. O. (1998). Controlled language - an introduction. In *Proceedings of CLAW ’98*, p.1-15.

Inria Atlas. (2005). KM3: *Kernel MetaMetaModel*. Technical report, LINA & INRIA, Nantes.

Isahara, H. (2013). Simplified natural language: Basic concepts and general principles - working survey in Japan. In *PROCEEDINGS OF LaRC*. http://www.unisa.ac.za/research/news/wp-content/uploads/2013/11/LaRC-2013-Proceedings-FINAL.pdf

Java Community Process (2002). Java(TM) Metadata Interface API specification 1.0 final release, JSR-000040.

James, M. (1967). *Design of Real-Time Computer Systems*. 1st Edition. Upper Saddle River, NJ, USA: Prentice-Hall, Inc., 629 p.

Hoard, J. E., Wojcik, R., Holzhauser, K. (1992). An automated grammar and style checker for writers of Simplified English. In *Computers and Writing: State of the Art,* Springer Netherlands, p. 278-296.

Jansen, B. J., Spink, A., Saracevic, T. (1998). Failure Analysis in Query Construction: Data and Analysis from a Large Sample of Web Queries. In *ACM DL,* p. 289-290.

Jarrar, M., Maria, C., Dongilli, K. P. (2006). *Multilingual verbalization of ORM conceptual models and axiomatized ontologies*. Technical report, STARLab, Vrije Universiteit Brussel.

McDermott, J. (1981). R1: The formative years. *AI Magazine*, 2(2), p. 21-29.

Shortliffe, E.H. (1976). *Computer Based Medical Consultations*. MYCIN, American Elsevier.

Johnson, J. (1995). Chaos: the dollar drain of IT project failures. *Application Development Trends*, 2, 1 p. 41-47.

Johnson, E. (2000). Talking across frontiers. In *Proceedings of the International Conference on European Cross Border Cooperation: Lessons for and from Ireland*.

Johnson, J. (2007). *CHAOS 2006 Research Project*. CHAOS Activity News 2.

Johnson, S. J. (1979). *Yacc: Yet Another Compiler Compiler, UNIX Programmer's Manual*, vol. 2, Holt, Rinehart, and Winston, New York, USA , p. 353–387.

Jouault, F. et Bézivin, J. (2006). KM3 : a DSL for Metamodel Specification. In *Proceedings of the IFIP International Conference on Formal Methods for Open Object-Based Distributed Systems (FMOODS)*, volume 4037 of Lecture Notes in Computer Science. Springer, p. 171–185.

Joubert, P., Kroeze, J. H., De Villiers, C. (2013). A grammar of business rules in Information Systems. *TD: The Journal for Transdisciplinary Research in Southern Africa*, 9(2), p. 241-276.

Jurafsky, D. et Martin, J. H. (2008). *Speech and Language Processing: An Introduction to Natural Language Processing, Speech Recognition, and Computational Linguistics*. 2nd ed., Prentice-Hall.

Kahlaoui, A. (2011). *Méthode pour la définition des langages dédiés basée sur le métamodèle ISO/IEC 24744*. Thèse de doctorat, École de technologie supérieure, Montréal, Canada.

Kaljurand, K. (2007). *Attempto Controlled English as a Semantic Web Language*. Ph.D. Thesis, Faculty of Mathematics and Computer Science, University of Tartu, Estonia.

Kaljurand, K. et Fuchs, N. E. (2006). Bidirectional mapping between OWL DL and Attempto Controlled English. In *Proceedings of PPSWR'06*, Springer Berlin Heidelberg, p. 179–189.

Kamp, H. et Reyle, U. (1993). *From discourse to logic: Introduction to model theoretic semantics of natural language, formal logic and discourse representation theory* (No. 42). Springer.

Kamprath, C., Adolphson, E., Mitamura, T. et al. (1998). Controlled language for multilingual document production: Experience with Caterpillar technical English. In *Proceedings of the Second International Workshop on Controlled Language Applications* (Vol. 146).

Kang, J. et Saint-Dizier, P. (2014). Towards an Error Correction Memory to Enhance Technical Texts Authoring in LELIE. In *Controlled Natural Language* (CNL 2014). Springer International Publishing, p. 55-65.

Khruathong, S. (2004). Une approche d'analyse syntaxique du thaï en Prolog en vue d'une traduction automatique ou assistée. In *proceedings of the workshop machine translation,*

controlled languages and specialised languages. Besançon, 5-6 May 2004, Lingvisticae Investigationes, Benjamins.

Kleppe, A., Warmer, J., Bast, W. (2003). *MDA Explained: The Model Driven Architecture: Practice and Promise*. Addison-Wesley Professional, ISBN: 032119442X, 192 p.

Kittredge, R. I. (2003). Sublanguages and controlled languages. In Ruslan Mitkov, editor, *The Oxford Handbook of Computational Linguistics*, p. 430–447.

Kuhn, T. (2007). Acerules: Executing rules in controlled natural language . In *proceedings of First International Conference on Web Reasoning and Rule Systems*. Springer Berlin Heidelberg, p. 299-308.

Kuhn, T. (2009). How controlled English can improve semantic wikis. In *proceedings of SemWiki2009: The Fourth Workshop on Semantic Wikis*.

Kuhn, T. (2010). *Controlled English for Knowledge Representation*. Doctoral Thesis, Faculty of Economics, Business Administration and Information Technology of the University of Zurich.

Kuhn, T. (2014). A Survey and Classification of Controlled Natural Languages. In *Computational Linguistics*, MIT Press Journal, Vol. 40, No. 1, p. 121-170.

Linehan, M. H. (2008). SBVR use cases. In R*ule Representation, Interchange and Reasoning on the Web*. Vol. 5321, Springer Berlin Heidelberg, Heidelberg, p. 182-196.

Lucie, B., Sophie, R., Bertrand, G. et al. (2006). *Report on State of the Art and Prospective Evolution of Formal Languages for Business Rules*. Public Research Centre Henri Tudor, Luxembourg.

Lukichev, S. et Wagner, G. (2006). Verbalization of the REWERSE I1 rule markup language. *Deliverable I1-D6, REWERSE*.

Marinos, A. et Krause, P. (2009). An SBVR Framework for RESTful Web Applications, In *International Symposium on RuleML 2009*. LNCS, Volume 5858, p.144-158.

Martin, P. (2002). Knowledge representation in CGLF, CGIF, KIF, frame-CG and formalized-english. In *Conceptual Structures: Integration and Interfaces*. Springer Berlin Heidelberg, p. 77-91.

Martin, J. (1967). *Design of Real-Time Computer Systems*. Upper Saddle River, 1st Edition NJ, USA: Prentice-Hall, Inc., 629 p.

Mateescu, A. et Salomaa, A. (1997). Formal languages: an introduction and a synopsis. In *Handbook of formal languages*. Springer Berlin Heidelberg, p. 1-39.

McCoy, D. W. et Plummer, D. C. (2006). *Defining, Cultivating and Measuring Enterprise Agility*, Gartner Research.

Means, L. et Godden, K. (1996). The Controlled Automotive Service Language (CASL) project. In *Proceedings of CLAW 1996*, p. 106–114.

Means, L. G., Chapman, P., Liu, A. (2000). Training for controlled language processes. In *Proceedings of CLAW 2000*, p.1–13.

Mitchell, R. L. (2006). Cobol: Not Dead Yet. *Computerworld*, *40*, 41. http://www.computerworld.com/article/2554103/app-development/cobol--not-dead-yet.html

Mitkov, R. (2005). *The Oxford Handbook of Computational Linguistics*. Oxford University Press, 806 p.

Moschoyiannis, S., Marinos, A., Krause, P. (2010). Generating SQL Queries from SBVR Rules, In *International Symposium on RuleML 2010*. LNCS 6403, Washington, DC, USA, October 21-23. Springer-Verlag Berlin Heidelberg, p. 128–143.

Monin, J.F. (2003). Understanding Formal Methods. Springer-Verlag, London.

Mueckstein, E. M. (1985). Controlled natural language interfaces: the best of three worlds. In *CSC '85: Proceedings of the 1985 ACM thirteenth annual conference on Computer Science*. ACM, p. 176-178.

Muegge, U. (2007). Controlled language: the next big thing in translation?. *ClientSide News Magazine*, 7(7), p. 21-24.

Muller, P. A. (2006). *De la modélisation objet des logiciels à la met modélisation des langages informatiques*. Mémoire d'Habilitation à Diriger les Recherches, Université de Rennes 1, Rennes.

Nebut, C. et Falleri, J.R. (2006). *Les transformations de modèles*. Technical report, LIRMM / Université de Montpellier 2.

Nicolae, O. et Wagner, G. (2008). Verbalising R2ML rules into SBVR. In *10th IEEE International Symposium on Symbolic and Numeric Algorithms for Scientific Computing*. Timisoara 26-29 Sept. p. 265-272.

Object Management Group (2003a). MDA Guide, v1.0.1. (omg/2003-06-01).

Object Management Group (2003b). Common Warehouse Metamodel (CWM) v1.1.

Object Management Group (2004). Business Rule Team, Business Semantics of Business Rules (BSBR).

Object Management Group (2005). Software Process Engineering Metamodel (SPEM) v1.1.

Object Management Group (2006a). Meta Object Facility (MOF) Core Specification, OMG Specification, v2.0. (formal/2006-01-01).

Object Management Group (2006b). Object Constraint Language (OCL), OMG Specification, v2.0. (formal/2006-05-01).

Object Management Group (2007). Unified Modeling Language (UML) 2.1.2 Superstructure, November 2007. Final Adopted Specification.

Object Management Group (2008a). Semantics of Business Vocabulary and Business Rules (SBVR), OMG Specification, v1.0. (formal/2008-01-02).

Object Management Group (2008b). Meta Object Facility (MOF) 2.0 Query/View/Transformation (QVT) Specification, version 1.0, April 2008.

Object Management Group (2008c). Business-Friendly Notation for Business Vocabulary and Rules. Request for Proposal. (bmi/2008-06-01).

Ogden, C. K. (1930). *Basic English: a general introduction with rules and grammar*.

Paul Treber & Co., London

Ogden, C. K. (1932). *The A B C of Basic English (in Basic)*. Psyche Miniatures General Series. K. Paul, Trench, Trubner, London.

Olivé, A. (2007). *Conceptual Modeling of Information Systems*. Springer-Verlag.

Papadopoulou E. et Portella P. M. (2009). Abduction Alerts in Greek and Spanish. In *ISMTCL Proceedings, International Review Bulag, PUFC*, p. 185-189.

Parr, T. J. et Quong, R. W. (1995). ANTLR: A predicated-LL (k) parser generator. *Software - Practice and Experience 25*, no. 7, p. 789–810.

Pereira, F. et Warren, D. (1986). Definite clause grammars for language analysis. In *Readings in natural language processing*. Morgan Kaufmann Publishers Inc, p. 101-124.

Pool, J. (2006). Can Controlled Languages scale to the Web? In *5th International Workshop on Controlled Language Applications*. Cambridge, Massachusetts, USA.

Projet Accord (2002). *La demarche MDA*. Technical report, Réseau National des Techniques Logicielles (RNTL), Mai 2002. Livrable 1.1-5.

Raj, A., Prabhakar, T. V., Hendryx, S. (2008). Transformation of SBVR business design to UML models. In *the proceedings of the 1st India software engineering conference*. ACM, p. 29-38.

Ram, S. et Khatri, V. (2005). A comprehensive framework for modeling set-based business rules during conceptual database design. *Information Systems*, 30(2), p. 89-118.

Rector, A., Drummond, N., Horridge, M. et al. (2004). OWL pizzas: Practical experience of teaching OWL-DL: Common errors & common patterns. In *Engineering Knowledge in the Age of the Semantic Web*. Springer Berlin Heidelberg, p. 63-81.

Renahy J. 2009. Controlled Languages: a Scientific Popularization through the Example of the Controlled Language ``LiSe''. In *ISMTCL Proceedings, International Review Bulag*, p. 215-222.

Renahy, J., Devitre, D., Dziadkiewicz, A. et al. (2009). Controlled language norms for the redaction of security protocols: finding the median between system needs and user acceptability. In *the 11th International Symposium on Social Communication*. Santiago de Cuba, Cuba.

Renahy, J., Gin, J., Devitre, D. et al. (2010). Development and evaluation of a controlled language and of a computerized writing assistant LiSe, to improve the quality and the safety of medical protocols. In *International Forum on Quality and Safety in Healthcare*, Nice.

Renahy, J., Thomas, I., Chippeaux G. et al. (2012). La « langue contrôlée » et l‟informatisation de son utilisation au service de la qualité des textes médicaux et de la sécurité dans le domaine de la santé. In *Systèmes d'information pour l'amélioration de la qualité en santé.* Springer Paris, p. 97-108.

Roche, C. (2003a). The differentia principle as a cornerstone of ontology. *Knowledge Management and Philosophy. Workshop in WM 2003 Conference*, Luzern, April 2- 4.

Roche, C. (2003b). Ontology: a survey. In *8th Symposium on automated systems based on Human Skill and Knowledge, IFAC*, Göteborg, Sweden, September 22-24.

Ross, R. G. (1997). *The Business Rule Book: Classifying, Defining and Modeling Rules*, Database. Research Group, Boston, MA, 2nd edition.

Ross, R. G. (2003). *Principles of the Business Rule Approach*. Boston, Addison-Wesley Longman Publishing Co., Inc.

Ross, R. G. (2005). *Business Rule Concepts, Getting to the Point of Knowledge.* (2nd ed.), Boston MA: Business Rule Solutions, LLC.

Ross, G. R. (2013). *Business Rule Concepts: Getting to the Point of Knowledge* (4e Ed), Business Rule Solutions, LLC, ISBN: 0-941049-14-0, 162 p.

Rudas, Z. (2009). Polish Controlled Language and Its Machine Translation into French. In *ISMTCL Proceedings, International Review Bulag*, *PUFC*, p. 231-235.

Ruffino, J. R. (1982). Coping with machine translation. In *Practical Experience of Machine Translation*. North-Holland Publishing Company, p. 57–60.

Rumbaugh, J., Jacobson, I. et Booch, G. (2004). *Unified Modeling Language Reference Manual*. 2nd ed., *The*. Pearson Higher Education.

Schactl, S. (1996). Requirements for Controlled German in Industrial Applications. In *Proceedings of the First International Workshop on Controlled Language Applications*. Katholieke Universiteit Leuven Centre for Computational Linguistics, p. 143-149.

Schwitter, R. (2002). English as a formal specification language. In *Proceedings of DEXA*. IEEE Computer Society, p. 228–232.

Schwitter, R., Ljungberg, A., Hood, D. (2003). ECOLE–A Look-ahead Editor for a Controlled Language. In *Proceedings of EAMT-CLAW03*, p. 141-150.

Schwitter, R. (2010). Controlled Natural Languages for Knowledge Representation. In *the 23rd International Conference on Computational Linguistics*, p. 1113-1121.

Sendall, S. et Kozaczynski, W. (2003). Model Transformation: The Heart and Soul of Model-Driven Software Development, *IEEE Software 20*, no. 5, p.42–45.

Skuce, D. (1975). An English-like language for qualitative scientific knowledge. In *the Fourth International Joint Conference on Artificial Intelligence*, p. 593-600.

Smart, J. M. (2003). Controlled English for global business. Writing for Translation. *The Guide from MultiLingual Computing & Technology*, (59), p.19-21

Sowa, J. F. 2000. *Controlled English*. http://www.jfsowa.com/logic/ace.htm

Sowa, J. F. (2004). *Common logic controlled english*. Technical report, 2004. Draft, 24 February 2004, http://www.jfsowa.com/clce/specs.htm.

Spaggiari, L., Beaujard, F., Cannesson, E. (2003). A controlled language at Airbus. In *Proceedings of EAMT-CLAW 2003*. Dublin City University, p. 151–159.

Standish (2004). *CHAOS*. The Standish Group International Inc. West Yarmouth, Massachusetts: Standish Group.

Stefik, M. (1995). *Introduction to knowledge systems*. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA.

Steinke, G. et Nickolette, C. (2003). Business rules as the basis of an organization‟s information systems. *Industrial Management & Data Systems*, 103, p. 52–63.

Steinberg, D., Budinsky, F., Paternostro, M. et al. (2008). EMF: Eclipse Modeling Framework, 2nd Edition, Addison-Wesley Professional, 744 p.

Steven, G. P. (1996). Controlled language for knowledge representation. In *Proceedings of the First International Workshop on Controlled Language Applications (CLAW 96)*. Katholieke Universiteit Leuven, Belgium, p. 233-242.

Strevens, P. et Johnson, E. (1983). SEASPEAK: A project in applied linguistics, language engineering, and eventually ESP for sailors. *ESP Journal*, 2 (2) Temnikova: p. 123-129.

Temnikova, I. et Margova, R. (2009). Towards a Controlled Language in Crisis Management: The Case of Bulgarian. In *Proceedings of the International Symposium on Data and Sense Mining, Machine Translation and Controlled Languages ISMTCL*. Besancon, France.

Temnikova, I. et Orasan, C. 2009. Post-editing Experiments with MT for a Controlled Language. Proceedings of the International Symposium on Data and Sense Mining, Machine Translation and Controlled Languages ISMTCL. Besancon, France, July 1-3.

Temnikova, I. (2012). *Text Complexity and Text Simplification in the Crisis Management Domain*. Ph.D. thesis, University of Wolverhampton, Wolverhampton.

Thongglin, K. (2014). *Controlled language for thai software requirements specification*. Thèse de doctorat, Centre Tesnière, Université de Franche-Comté, Besançon.

US Government Accountability Office (2006). *Financial Management Systems-Additional Efforts Needed to Address Key Causes of Modernization Failures*. Report to Congressional Requesters. GAO-06-184. http://www.gao.gov/assets/250/249279.pdf.

Vassiliou, M., Markantonatou, S., Maistros, Y. et al. (2003). Evaluating Specifications for Controlled Greek", EAMT-CLAW2003, Dublin, Ireland

Vincent, P. et Isaac, F. (2005). Fair Isaac Blaze Advisor Structured Rules Language - a commercial rules representation. In *W3C Workshop on Rule Languages for Interoperability*.

Voice of America (2009). VOA *Special English Word Book: a list of words used in Special*

English programs on radio, television, and the Internet.

Von Halle, B. (2001). *Business rules applied: building better systems using the business rules approach.* John Wiley & Sons, Inc, 592 p.

Von Halle, B. et Goldberg, L. (2006). *The Business Rule Revolution (ebook): Running Business the Right Way*, von Halle & L. Goldberg eds, Happy About.

Wagner, G., Lukichev, S., Fuchs, N. E. et al. S. (2005). First-version controlled english rule language. *Deliverable I1-D2, REWERSE IST, 506779.*

Waltz, D. L. (1978). An English language question answering system for a large relational database. *Communications of the ACM*, 21(7), p. 526-539.

White, C. et Schwitter, R. (2009). An update on PENG light. In *Proceedings of ALTA.* Vol. 7, p. 80-88.

Wojcik, R. H. et James E. Hoard. (1997). Controlled languages in industry. In *Survey of the state of the art in human language technology*. Cambridge University Press, p. 238–239.

Wu, X., Cardey, S., Greenfield, P. (2004). Designing a controlled language for the machine translation of medical protocols: The case of English to Chinese, In *Machine Translation: From Real Users to Research*. Springer Berlin Heidelberg, p. 37-47.

Wyner, A., Angelov, K., Barzdins, G. et al. (2009). On Controlled Natural Languages: Properties and Prospects. In *Controlled Natural Language (CNL 2009)*. Springer Berlin Heidelberg, p. 281-289.

Zhang, W. et Yu, S.W. (1998). Construction of Controlled Chinese Lexicon. In *Proceedings of the Second International Workshop on Controlled Language Applications CLAW98*. Pittsburgh, Pennsylvania: Language Technologies Institute, Carnegie Mellon University, p. 159-173.

Annexes

I **Manifeste pour les règles métier** (BRG, 2003).

Business Rules Manifesto

The Principles of Rule Independence

by Business Rules Group

Article 1. *Primary Requirements, Not Secondary*

1.1. Rules are a first-class citizen of the requirements world.

1.2. Rules are essential for, and a discrete part of, business models and technology models.

Article 2. *Separate From Processes, Not Contained In Them*

2.1. Rules are explicit constraints on behavior and/or provide support to behavior.

2.2. Rules are not process and not procedure. They should not be contained in either of these.

2.3. Rules apply *across* processes and procedures. There should be one cohesive body of rules, enforced consistently across all relevant areas of business activity.

Article 3. *Deliberate Knowledge, Not A By-Product*

3.1. Rules build on facts, and facts build on concepts as expressed by terms.

3.2. Terms express business concepts; facts make assertions about these concepts; rules constrain and support these facts.

3.3. Rules must be explicit. No rule is ever assumed about any concept or fact.

3.4. Rules are basic to what the business knows about itself — that is, to basic business knowledge.

3.5. Rules need to be nurtured, protected, and managed.

Article 4. *Declarative, Not Procedural*

4.1. Rules should be expressed declaratively in natural-language sentences for the business audience.

4.2. If something cannot be expressed, then it is not a rule.

4.3. A set of statements is declarative only if the set has no implicit sequencing.

4.4. Any statements of rules that require constructs other than terms and facts imply assumptions about a system implementation.

4.5. A rule is distinct from any enforcement defined for it. A rule and its enforcement are separate concerns.

4.6. Rules should be defined independently of responsibility for the *who*, *where*, *when*, or *how* of their enforcement.

4.7. Exceptions to rules are expressed by other rules.

Article 5. *Well-Formed Expression, Not Ad Hoc*

5.1. Business rules should be expressed in such a way that they can be validated for correctness by business people.

5.2. Business rules should be expressed in such a way that they can be verified against each other for consistency.

5.3. Formal logics, such as predicate logic, are fundamental to well-formed expression of rules in business terms, as well as to the technologies that implement business rules.

continued…

Article 6. *Rule-Based Architecture, Not Indirect Implementation*

6.1. A business rules application is intentionally built to accommodate continuous change in business rules. The platform on which the application runs should support such continuous change.

6.2. Executing rules directly – for example in a rules engine – is a better implementation strategy than transcribing the rules into some procedural form.

6.3. A business rule system must always be able to explain the reasoning by which it arrives at conclusions or takes action.

6.4. Rules are based on truth values. How a rule's truth value is determined or maintained is hidden from users.

6.5. The relationship between events and rules is generally many-to-many.

Article 7. *Rule-Guided Processes, Not Exception-Based Programming*

7.1. Rules define the boundary between acceptable and unacceptable business activity.

7.2. Rules often require special or selective handling of detected violations. Such rule violation activity is activity like any other activity.

7.3. To ensure maximum consistency and reusability, the handling of unacceptable business activity should be separable from the handling of acceptable business activity.

Article 8. *For the Sake of the Business, Not Technology*

8.1. Rules are about business practice and guidance; therefore, rules are motivated by business goals and objectives and are shaped by various influences.

8.2. Rules always cost the business something.

8.3. The cost of rule enforcement must be balanced against business risks, and against business opportunities that might otherwise be lost.

8.4. 'More rules' is not better. Usually fewer 'good rules' is better.

8.5. An effective system can be based on a small number of rules. Additional, more discriminating rules can be subsequently added, so that over time the system becomes smarter.

Article 9. *Of, By and For Business People, Not IT People*

9.1. Rules should arise from knowledgeable business people.

9.2. Business people should have tools available to help them formulate, validate and manage rules.

9.3. Business people should have tools available to help them verify business rules against each other for consistency.

Article 10. *Managing Business Logic, Not Hardware/Software Platforms*

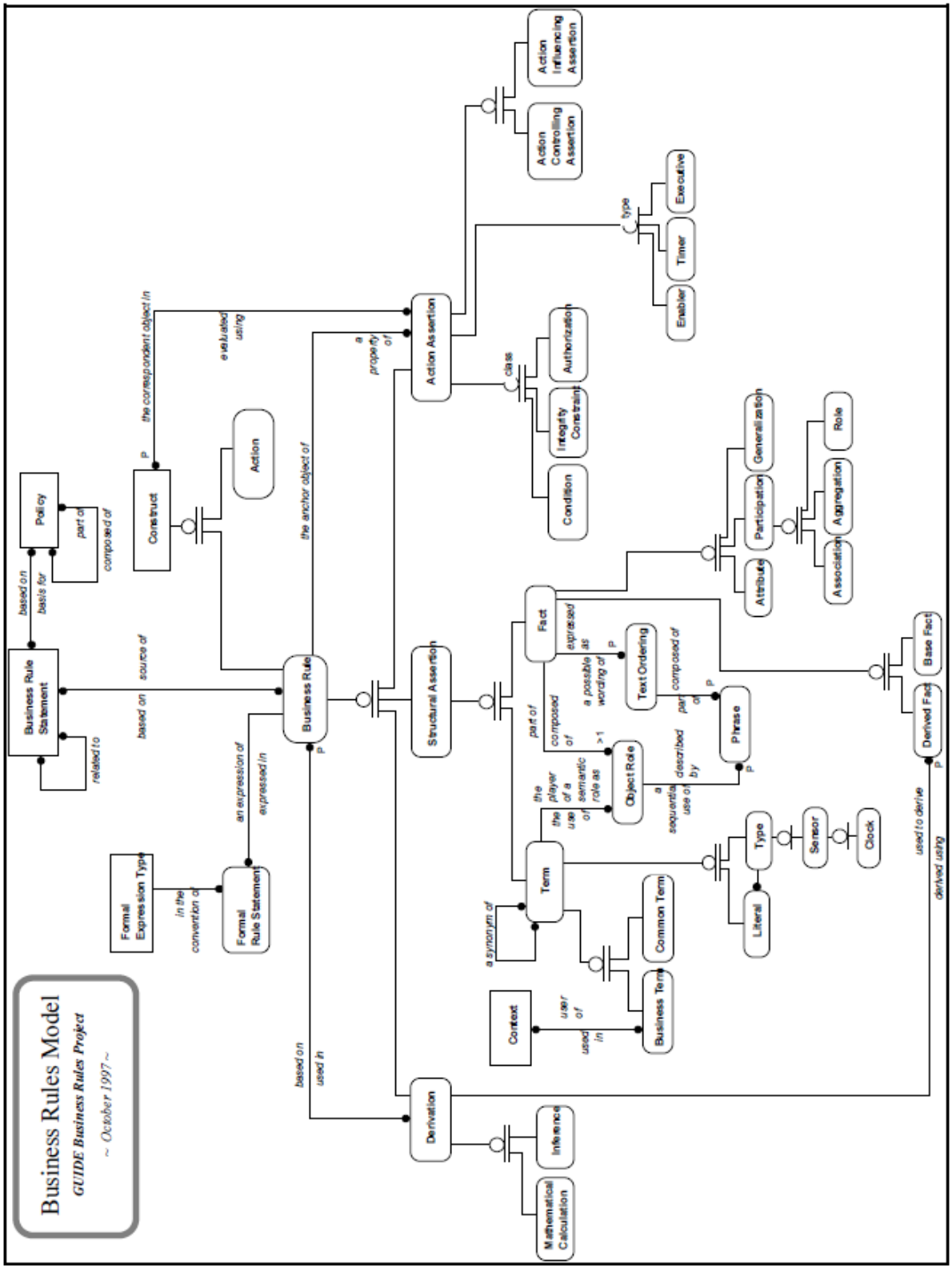
10.1. Business rules are a vital business asset.

10.2. In the long run, rules are more important to the business than hardware/software platforms.

10.3. Business rules should be organized and stored in such a way that they can be readily redeployed to new hardware/software platforms.

10.4. Rules, and the ability to change them effectively, are fundamental to improving business adaptability.

II *The Complete Business Rules Model* (BRG, 2000).



III The timeline of the evolution of controlled English (Kuhn, 2014).