Construction d'ontologies à partir de textes : une approche basée sur les transformations de modèles

Henry Valery Teguiak

Ecole Nationale Supérieure de Mécanique et d'Aérotechnique
Laboratoire d'Informatique et d'Automatique pour les Systèmes

THESE

pour l'obtention du Grade de

DOCTEUR DE L'ÉCOLE NATIONALE SUPÉRIEURE DE MÉCANIQUE ET D'AEROTECHNIQUE

(Diplôme National – Arrêté du 7 août 2006)

Ecole Doctorale : Sciences et Ingénierie pour l'Information, Mathématiques
Secteur de Recherche : INFORMATIQUE ET APPLICATIONS

Présentée par :

Henry Valéry TEGUIAK

****

Construction d'ontologies à partir de textes : une approche basée sur les transformations de modèles

****

Directeurs de Thèse : **Guy PIERRA, Yamine AIT-AMEUR** et **Ladjel BELLATRECHE**

Soutenue le 12 Décembre 2012
devant la Commission d'Examen

JURY

Président :	**Jean CHARLET**	Chargé de Recherche/Professeur associé, Ecole Centrale Paris
Rapporteurs :	**Djamal BENSLIMANE**	Professeur, Université Claude Bernard / LIRIS, Lyon
	Sylvie DESPRES	Professeur, Université de Paris XIII / Lim&Bio, Paris
Examinateurs :	**Yamine AIT-AMEUR**	Professeur, INPT-ENSEEIHT / IRIT, Toulouse
	Ladjel BELLATRECHE	Professeur, ISAE-ENSMA / LIAS, Futuroscope
	Guy PIERRA	Professeur, LIAS, Futuroscope
	Eric SARDET	Directeur Technique, CRITT Informatique, Futuroscope

Merci à

■ Pr. Guy PIERRA, qui a guidé mes premiers pas dans le monde de la recherche et sans qui ce travail n'aurait jamais vu le jour. Je lui souhaite par la même occasion beaucoup de courage face à la situation qu'il traverse en ce moment ;

■ Pr. Yamine AIT-AMEUR, pour m'avoir guidé pendant ces années de thèse. Je le remercie d'avoir eu une oreille attentive à certains choix qui me tenaient à cœur ;

■ Pr. Ladjel BELLATRECHE qui, au-delà de l'encadrement de thèse, m'aura prodigué beaucoup de conseils ;

■ Dr. Éric SARDET pour toute sa disponibilité à m'aider pendant cette thèse. Qu'il trouve ici l'expression de ma profonde reconnaissance pour l'expérience professionnelle que j'ai acquise à ses cotés durant mes années de travail au CRITT Informatique ;

■ Pr. Jean CHARLET qui, en plus d'avoir facilité mon intégration au sein du projet DaFOE4App, m'a fait l'honneur de présider le jury de ma thèse ;

■ Pr. Djamal BENSLIMANE et Pr. Sylvie DESPRES de m'avoir fait l'honneur d'être rapporteurs de cette thèse et à qui je souhaite exprimer ma profonde reconnaissance ;

■ aux partenaires du projet DaFOE4App (Inserm, IRIT, LIAS, LIPN, MONDECA, Supélec, Télécom Paris Tech,UTC) pour toutes les discussions constructives que nous avons eues ;

■ Pr. Francis COTTET et Pr. Emmanuel GROLLEAU pour leur aide face aux problèmes administratifs que j'ai connus durant la fin de ce travail ;

■ mes collègues du LIAS pour l'ambiance que nous avons partagée. Je pense en particulier à Mickaël BARON qui comme moi est un passionné des technologies Java/JEE, Stéphane JEAN pour sa relecture, Frédéric CARREAU et Claudine RAULT pour leurs aides techniques et administratives respectives;

■ Dago AGBODAN, François DORIN, Jean-Claude POTIER et Guillaume TEXIER, mes anciens collègues du CRITT Informatique, avec qui j'ai partagé une expérience industrielle enrichissante ;

■ mon papa Abraham KAMMENGNE, à qui je dois toute ma détermination ;

■ ma famille pour le soutien sans faille et les encouragements qu'elle m'a apportés. Un merci particulier à mes frères KOUOGANG NKRUMAH et Justin-Hervé NOUBISSI pour toutes les remarques et suggestions qu'ils m'ont faites après avoir lu cette thèse ;

■ aux familles KAMDEM, POIKA et TAKOUAM pour leur soutien sans cesse renouvelé ;

■ Morelle FANKOU pour le soutien qu'elle a su m'accorder ;

■ mes ami(e)s Nabil BELAID, Félix BOGNOU, Hérvé CHOMENI, Hondjack DEHAINSALA, Chimène FANKAM, Ferdinand FOTIE, Christian FOTSING, Francis GOUPEYOU, Benoît GUEFANO, Georges KEMAYO, Gabriel KEMJE, Jean-Claude LAOUWAYI, Gabriel NGADOU, Anne-Lise NJONJO, Stéphane NJONJO, Célanie NOUIND, André-Parfait NYEMECK, Alphonse-Marie ONAMBELE, Fred ONANINA, Laura ONDOUA, Ferdinand OWOUNDI, Olivier PIEUME, Célestin TALOM, Dieudonné TCHUENTE, Franc TETANG et Raoul TIAM pour leurs encouragements.

■ enfin à Dieu Tout Puissant qui m'a donné la force d'aller jusqu'au bout de ce travail.

Épigraphe

"A pessimist sees the difficulty in every opportunity;
an optimist sees the opportunity in every difficulty."
– Winston Churchill

A la mémoire de ma mère.

Table des matières

Résumé

Depuis son émergence au début des années 1990, la notion d'ontologie s'est rapidement diffusée dans un grand nombre de domaines de recherche. Compte tenu du caractère prometteur de cette notion, de nombreux travaux portent sur l'utilisation des ontologies dans des domaines aussi divers que la recherche d'information, le commerce électronique, le web sémantique, l'intégration de données, etc.. L'efficacité de tous ces travaux présuppose l'existence d'une ontologie de domaine susceptible d'être utilisée. Or, la conception d'une telle ontologie s'avère particulièrement difficile si l'on souhaite qu'elle fasse l'objet de consensus. S'il existe des outils utilisés pour éditer une ontologie supposée déjà conçue, et s'il existe également plusieurs plate-formes de traitement automatique de la langue permettant d'analyser automatiquement les corpus et de les annoter tant du point de vue syntaxique que statistique, il est difficile de trouver une procédure globalement acceptée, ni a fortiori un ensemble d'outils supports permettant de concevoir une ontologie de domaine de façon progressive, explicite et traçable à partir d'un ensemble de ressources informationnelles relevant de ce domaine. L'objectif du projet ANR DaFOE4App (Differential and Formal Ontologies Editor for Application), dans lequel s'inscrit notre travail, était de favoriser l'émergence d'un tel ensemble d'outils. Contrairement à d'autres outils de construction d'ontologies, la plate-forme DaFOE, présentée dans cette thèse, ne propose pas un processus de construction figé ni en nombre d'étapes, ni sur la représentation des étapes. En effet, dans cette thèse nous généralisons le processus de construction d'ontologies pour un nombre quelconque d'étapes. L'intérêt d'une telle généralisation étant, par exemple, d'offrir la possibilité de raffiner le processus de construction en insérant ou modifiant des étapes. On peut également souhaiter supprimer certaines étapes à fin de simplifier le processus de construction. L'objectif de cette généralisation est de minimiser l'impact de l'ajout, suppression ou modification d'une étape dans le processus global de construction d'ontologies, tout en préservant la cohérence globale du processus de construction. Pour y parvenir, notre approche consiste à utiliser l'Ingénierie Dirigée par les Modèles pour caractériser chaque étape au sein d'un modèle et ensuite ramener le problème du passage d'une étape à l'autre à un problème de mapping de modèles. Les mappings établis entre les modèles sont ensuite utilisés pour semi-automatiser le processus de construction d'ontologies. Ce processus de construction se faisant dans un contexte persistant de base de données, nous proposons dans cette thèse, d'une part, pour les bases de données dites à base de modèles (BDBM) du fait qu'elles permettent de stocker à la fois les données et les modèles décrivant ces données, une extension pour la prise en compte des mappings, et, d'autre part, nous proposons le langage de requête MQL (Mapping Query Language) qui, en masquant la complexité de l'architecture de la BDBM facilite son exploitation. L'originalité du langage MQL se trouve dans sa capacité, au travers de requêtes syntaxiquement compactes, à explorer transitivement tout ou partie du graphe de mappings lors d'une recherche d'informations.

Mots-clés : Base de données à base de modèles, Langage de requête, Mappings, Méta-modélisation, Ontologie, Transformation de modèles.

Abstract

Since its emergence in the early 1990s, the notion of ontology has been quickly distributed in many areas of research. Given the promise of this concept, many studies focus on the use of ontologies in many areas like information retrieval, electronic commerce, semantic Web, data integration, etc.. The effectiveness of all this work is based on the assumption of the existence of a domain ontology that is already built an that can be used. However, the design of such ontology is particularly difficult if you want it to be built in a consensual way. If there are tools for editing ontologies that are supposed to be already designed, and if there are also several platforms for natural language processing able to automatically analyze corpus of texts and annotate them syntactically and statistically, it is difficult to find a globally accepted procedure useful to develop a domain ontology in a progressive, explicit and traceable manner using a set of information resources within this area. The goal of ANR DaFOE4App (Differential and Formal Ontology Editor for Application) project, within which our work belongs to, was to promote the emergence of such a set of tools. Unlike other tools for ontologies development, the platform DaFOE presented in this thesis does not propose a methodology based on a fixed number of steps with a fixed representation of theses steps. Indeed, in this thesis we generalize the process of ontologies development for any number of steps. The interest of such a generalization is, for example, to offer the possibility to refine the development process by inserting or modifying steps. We may also wish to remove some steps in order to simplify the development process. The aim of this generalization is for instance, for the overall process of ontologies development, to minimize the impact of adding, deleting, or modifying a step while maintaining the overall consistency of the development process. To achieve this, our approach is to use Model Driven Engineering to characterize each step through a model and then reduce the problem of switching from one step to another to a problem of models transformation. Established mappings between models are then used to semi-automate the process of ontologies development. As all this process is stored in a database, we propose in this thesis, for Model Based Database (MBDB) because they can store both data and models describing these data, an extension for handling mappings. We also propose the query language named MQL (Mapping Query Language) in order to hide the complexity of the MBDB structure. The originality of the MQL language lies in its ability, through queries syntactically compact, to explore the graph of mappings using the transitivity property of mappings when retrieving informations.

Keywords : Mappings, Metamodelling, Model-based Database, Model transformation, Ontology, Query language.

Introduction Générale

Contexte

Depuis son émergence au début des années 1990, dans les recherches en modélisation de connaissances, la notion d'ontologie s'est rapidement diffusée dans un grand nombre de domaines de recherche en informatique. Définie comme la représentation formelle et consensuelle au sein d'une communauté d'utilisateurs, des concepts propres à un domaine et des relations qui les relient, la notion d'ontologie apparaît comme un moyen de représenter explicitement et de partager des objets d'un domaine ainsi que leur sémantique. Compte tenu du caractère prometteur de cette notion, de nombreux travaux portent sur l'utilisation des ontologies dans des domaines aussi divers que le traitement automatique de la langue naturelle, la recherche d'information, le commerce électronique, le web sémantique, la spécification des composants logiciels, l'intégration de systèmes d'information, etc..

L'efficacité de tous ces travaux présuppose l'existence d'une ontologie de domaine susceptible d'être utilisée. Or, la conception d'une telle ontologie s'avère particulièrement difficile si l'on souhaite qu'elle fasse l'objet de consensus dans une communauté assez large. Un moyen très largement utilisé pour atteindre cet objectif est de partir d'éléments préexistants dans le domaine : corpus textuels, taxonomies, fragments d'ontologies préexistants, des schémas de bases de données, etc. et de les exploiter comme connaissance *a priori* pour la construction progressive d'une ontologie du domaine. Cette tâche correspond à un apport de connaissances et est difficilement automatisable. Dans le cas de la construction d'ontologies à partir de textes, par exemple, il existe néanmoins, et en particulier lorsque des corpus importants sont utilisés, la possibilité de recourir à des outils informatiques pour faciliter l'extraction des *termes* et des *relations*, l'analyse syntaxique et distributionnelle, l'identification des synonymes et homonymes, etc., et cela, jusqu'à la représentation formelle de l'ontologie.

Par ailleurs, s'il existe des outils tels que Protégé [1], PLibEditor [2], OntoEdit, etc., utilisés pour éditer formellement une ontologie supposée déjà conçue, et s'il existe également plusieurs outils de traitement automatique de la langue (TAL) permettant d'analyser automatiquement les corpus de textes et de les annoter sur les points de vue syntaxique, distributionnel et statistique, il est difficile de trouver une procédure globalement acceptée, ni a fortiori un ensemble d'outils supports permettant de concevoir une ontologie de domaine de façon progressive, explicite et traçable à partir d'un ensemble de ressources

1. http://protege.stanford.edu/
2. http://www.plib.ensma.fr/

informationnelles relevant de ce domaine.
Pour favoriser l'émergence d'un tel ensemble d'outils, le projet DaFOE4App [3] a été initié en 2006 avec comme objectifs de :

- $\mathbf{O}_1$) définir un cadre méthodologique général susceptible d'intégrer la réalisation des scénarios très divers de conception d'ontologies ;
- $\mathbf{O}_2$) définir une structure de modélisation permettant de s'adapter à ces différents scénarios de conception ;
- $\mathbf{O}_3$) spécifier puis développer une plate-forme (la plate-forme DaFOE) capable d'accueillir et d'intégrer les différentes catégories d'outils actuellement utilisés de façon autonomes au sein d'une unique plate-forme, assurant à la fois la persistance et la traçabilité, et permettant, par exemple, d'aller de l'analyse d'un corpus, supposé annoté par un outil de TAL, à la définition d'une ontologie de domaine.

Problématique

Pour atteindre l'objectif $\mathbf{O}_1$ du projet DaFOE4App, une analyse préliminaire de besoins, initiée par le consortium du projet DaFOE4App, et visant à unifier les démarches d'élaboration d'une ontologie a permis de construire le processus de la figure 1. Cette analyse de besoins recouvre diverses méthodes de construction d'ontologies telles que la construction à partir de textes, du schéma d'une base de données, d'un thésaurus, d'une ontologie existante, etc.. Elle ne définit pas le processus de construction mais s'efforce de représenter les tâches usuelles (extraction de données terminologique, construction du réseau conceptuel, modélisation, etc.) ainsi que certaines représentations intermédiaires fréquemment utilisées.

De ce fait, cette analyse ne peut être interprétée directement comme définissant le processus de construction d'une ontologie. Néanmoins, cette représentation a permis de mettre en évidence certaines ambiguïtés telles que :

1. la représentation utilisée laisse penser que les différentes méthodes commencent au même endroit et aboutissent toutes au même endroit. Or, en réalité, ce n'est pas toujours le cas. Par exemple, il n'y a pas de données terminologiques à extraire si l'on part d'ontologies déjà constituées ;
2. un nom de tâche peut représenter des traitements très différents, les traitements à partir de textes n'étant par exemple pas les mêmes que ceux à partir des thésaurus ;
3. une nouvelle démarche de construction pourrait éventuellement nécessiter des tâches qui n'ont pas été prévues, auquel cas il faudrait ajouter ces tâches ;
4. etc.

De ce fait, une approche possible consisterait à :

- identifier, pour l'ensemble des méthodes, des étapes clairement définissables ;
- formaliser, pour chaque étapes le modèle correspondant ;
- fournir les points d'entrée et de sortie de chaque étape afin de s'adapter à des processus partiels ;
- permettre l'adjonction de nouveaux traitements utilisables pour passer d'une étape à l'autre.

3. Differential and Formal Ontologies Editor for Application: http://www.http://dafoe4app.fr/

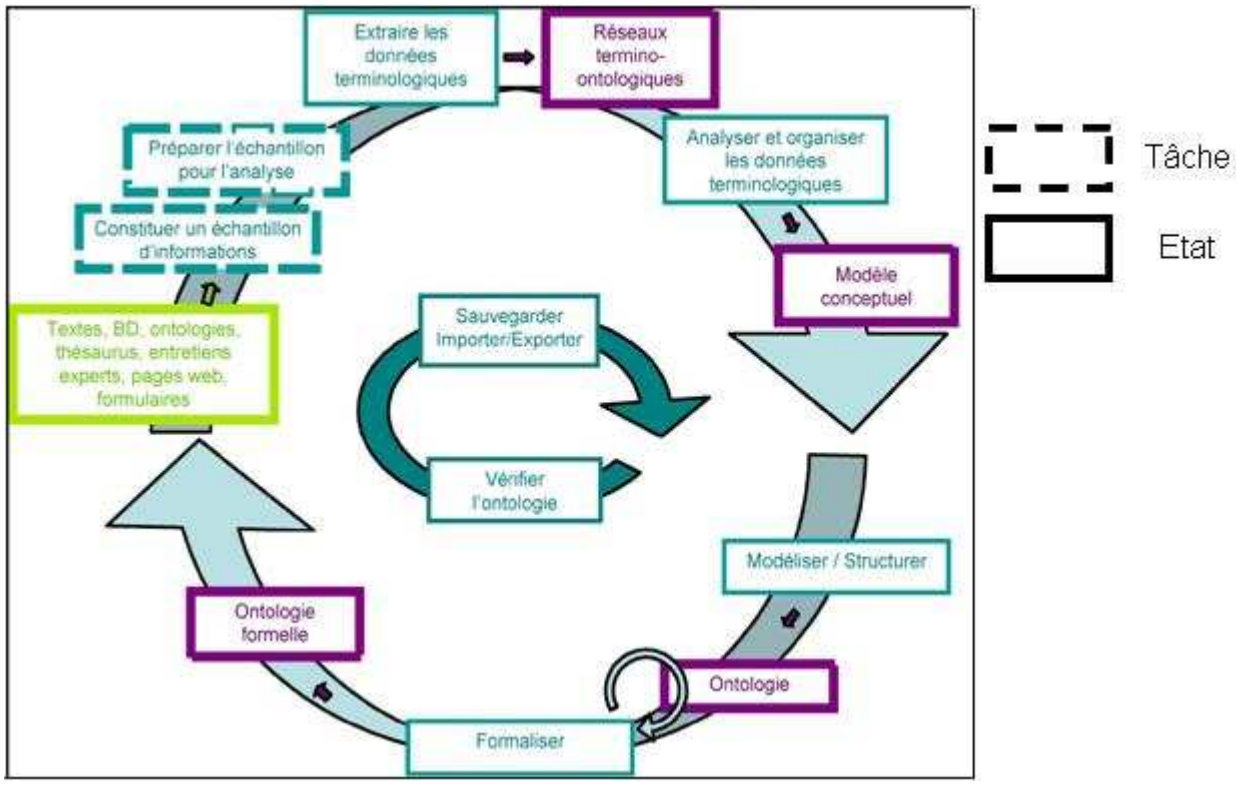

FIGURE 1 – Analyse préliminaire.

Ainsi, les trois objectifs du projet DaFOE4App peuvent être respectivement reprécisés sous forme d'exigences pour l'architecture support de la plate-forme à savoir :

- $\mathbf{E}_1$) définir clairement un certain nombre d'étapes utilisables par les différentes méthodes de construction d'ontologies ;
- $\mathbf{E}_2$) permettre certaines modifications pour les modèles de représentation utilisables dans les différentes étapes ;
- $\mathbf{E}_3$) permettre l'insertion, dans la plate-forme, de nouveaux traitements utilisables pour passer d'une étape à une autre.

Comme solution à l'exigence $\mathbf{E}_1$, le consortium du projet DaFOE4App propose un processus de construction en trois principales étapes d'analyse :

- une étape d'analyse terminologique qui permet de représenter les informations extraites par des outils de TAL ;
- une étape d'analyse termino-ontologique permettant de désambiguïser les informations issues de l'étape terminologique pour construire un réseau conceptuel ;
- une étape d'analyse ontologique permettant de formaliser les concepts dans un modèle d'ontologies.

Compte tenu d'une part, de la diversité des domaines des différents acteurs (ingénierie de la connaissance, linguistique, etc.) impliqués dans la démarche de construction d'ontologies à partir de textes, et, d'autre part, de la diversité des tâches ou traitements à effectuer pour passer d'une étape à l'autre, il est

difficile de représenter les différentes étapes de ce processus au sein d'un unique modèle. De ce fait, l'exigence $\mathbf{E}_2$ vise donc à la fois à accorder assez d'autonomie à chacune des étapes mais aussi à garantir la cohérence des données lors du passage d'une étape à l'autre, même en cas d'évolution du modèle d'une étape.

L'exigence $\mathbf{E}_3$, pour sa part, met un accent particulier sur la description du passage d'une étape à l'autre. En effet, à la différence des outils actuels de construction d'ontologies, la plate-forme DaFOE vise à offrir, à un utilisateur expérimenté en matière de la modélisation du processus de construction d'ontologies, la possibilité de contribuer à la description des règles de passage d'une étape à l'autre.

Proposition

En prenant en compte, d'une part, le besoin de pouvoir ajouter, modifier ou supprimer des tâches (et donc l'étape de construction correspondante), et, d'autre part, la diversité des domaines des différents acteurs (ingénierie de la connaissance, linguistique, etc.) impliqués dans la démarche de construction d'ontologies à partir de textes, il devient difficile de représenter les différentes étapes d'un tel processus de construction d'ontologies au sein d'un unique modèle. A cet effet, nous proposons dans cette thèse :

1. de généraliser cette construction dans le cas d'un processus en n étapes ($n \geq 2$). L'intérêt d'une telle généralisation étant, par exemple, d'offrir la possibilité de raffiner le processus de construction en insérant des étapes supplémentaires. On peut également souhaiter supprimer certaines étapes à fin de simplifier le processus de construction. Le challenge de cette généralisation est de minimiser l'impact de l'ajout, suppression ou modification d'une étape dans le processus global de construction d'ontologies, tout en préservant la cohérence globale de ce processus ;

2. d'utiliser les approches d'Ingénierie Dirigée par les Modèles (IDM) pour caractériser chaque étape au sein d'un modèle et ensuite ramener le problème du passage d'une étape à l'autre à un problème de transformation de modèles ;

3. d'accroître les performances en s'appuyant sur les bases de données pour la gestion de forte volumétrie. Ce besoin nous impose d'envisager des solutions pour la gestion des transformations dans un environnement persistant comme les bases de données relationnelles ;

4. d'élaborer une approche d'interrogation des modèles incluant un raisonnement sur les transformations existants entre les modèles.

A coté des propositions ci-dessus mentionnées, il existe d'autres propositions techniques liées notamment à l'architecture technique de la plate-forme DaFOE. La plus fondamentale de ces propositions étant de disposer d'une architecture extensible facilitant l'ajout de nouvelles fonctionnalités à la plate-forme DaFOE.

Organisation du mémoire

Dans la première partie de ce mémoire, nous présentons un état de l'art sur les problématiques autour de la construction d'ontologies et des transformations de modèles ainsi que de problèmes liés à la gestion des transformations dans les bases de données. La seconde partie, quant à elle, nous permettra, d'une part, de présenter nos propositions pour la modélisation des transformations en s'intéressant au cas particulier des bases de données, et, d'autre part, nous proposons une approche d'exploitation de ces transformations pour la gestion des données contenues dans la base de données. Cette partie présente également l'application de ces propositions dans le cadre de la construction d'ontologies suivant la démarche adoptée dans le projet DaFOE4App. Nous y présentons également les solutions techniques que nous avons proposées pour le développement de la plate-forme DaFOE.

Partie 1: État de l'art

Le chapitre 1 de cette partie présente les approches de construction d'ontologies ainsi que les outils actuellement disponibles. Ce chapitre permet de nous positionner en faveur d'un outil impliquant davantage l'utilisateur final (éventuellement expert) dans la démarche de construction. Nous montrons notamment que ce dernier, en spécifiant des transformations entre les modèles associés à chaque étape, influe sur le processus global de construction d'ontologies. Pour l'émergence d'un tel outil nous présentons notre approche dans laquelle il est possible de définir des transformations permettant de passer d'une étape à un autre. Dans le chapitre 2 nous illustrons l'Ingénierie Dirigée par le Modèles dans un contexte d'hétérogénéité des modèles en présentant le problème de la transformation de modèles. Nous mettons tout cela dans le contexte des bases de données en présentant les propositions fournies par la littérature. Tout ceci permet de dégager le besoin de réaliser des transformations de modèles en environnement persistant. A cet effet, nous discutons également de la capacité des architectures actuelles de base de données à supporter la persistance des transformations.

Partie 2: Contribution

Dans le chapitre 3 de cette partie, nous présentons notre représentation des modèles et des transformations. Les principales contributions de ce chapitre sont :

- l'extensibilité dynamique de la représentation des transformations grâce à la possibilité de créer de nouveaux constructeurs de mappings ;
- la formalisation de la gestion (interrogation, modification, etc.) d'un modèle en utilisant un autre modèle et la transformation entre ces modèles.

Dans le chapitre 4, nous présentons notre approche de structuration des bases de données permettant, d'une part, de persister notre représentation des modèles et des transformations, et, d'autre part, de supporter le passage à l'échelle. Enfin dans le chapitre 5, nous nous intéressons au problème de l'exploitation des transformations pour les opérations classiques sur les bases de données telles que l'interrogation, l'insertion, la suppression et la mise à jour. Nous présentons notamment le concept *d'instances ou données à base de mappings* que nous avons introduit pour caractériser les données extraites en exploitant les transformations entre les modèles. Le langage de requêtes MQL (Mapping Query Language), extension

du langage de requête SQL et intégrant des clauses dites *orientées mapping* permettant d'extraire ces *instances à base de mappings* est également présenté dans ce chapitre. Nous présentons la sémantique de chacune de ces clauses et expliquons, grâce à l'algèbre *MapAlgebra* que nous avons proposée, le principe de propagation des requêtes MQL d'un modèle à un autre.

Le chapitre 6 de cette partie est essentiellement consacré à l'implémentation de la plate-forme DaFOE et présente une visite guidée de celle-ci. Il présente nos propositions en réponse aux exigences techniques de la plate-forme. Nous y discutons, par exemple, du choix d'une architecture logicielle offrant assez de souplesse pour l'extensibilité de la plate-forme DaFOE. Nous présentons également notre modélisation de l'extensibilité de la plate-forme DaFOE et l'illustrons en décrivant un module contributeur permettant d'étendre la plate-forme DaFOE. Enfin, nous donnons des détails techniques aussi bien sur l'implémentation du langage de requêtes MQL que sur la conception de MQLToolkit, l'environnement dédié à la manipulation des requêtes MQL.

Ce manuscrit se termine par une conclusion générale dans laquelle nous rappelons la problématique de notre thèse ainsi que les différentes propositions formulées. Nous discutons également des insuffisances de nos travaux et formulons en guise de pistes de réflexion, quelques perspectives envisageables à la suite de nos travaux.

Première partie

État de l’art

"Comprendre, ce n’est pas tout comprendre,
c’est aussi reconnaître qu’il y a de l’incompréhensible."
– Edgar Morin

Chapitre 1

Construction d'ontologies

Sommaire

Résumé. En ingénierie des connaissances, les données peuvent être représentées suivant une approche informelle en utilisant le langage naturel (des textes décrivant le vocabulaire d'un domaine par exemple), ou suivant une approche semi-formelle au travers de modèles comme les thésaurus, ou encore par des modèles formels comme les ontologies. Ce chapitre présente ces trois niveaux de représentation à travers l'étude du processus de construction d'ontologies à partir de textes en décrivant les éléments intervenant dans la conception d'une ontologie depuis l'extraction de données textuelles à l'aide d'outils de traitements automatique de la langue jusqu'à la formalisation des concepts de l'ontologie.

1.1 Introduction

Depuis leur émergence, les ontologies apparaissent comme une solution à de nombreux problèmes tels que l'intégration de données, l'interopérabilité logicielle, le partage de données, etc.. De nombreuses méthodologies de construction d'ontologies ont été proposées ces dernières années. Ces méthodologies peuvent être classifiées en fonction de l'utilisation ou non de connaissances *a priori* ainsi que de techniques d'apprentissage. Les premières méthodologies, essentiellement manuelles, visent à construire des ontologies sans connaissance *a priori*. Si d'autres travaux se sont intéressés à la construction coopérative d'ontologies, très peu d'outils intègrent cette approche collaborative pourtant nécessaire dans le contexte des ontologies du fait qu'elle permettrait une construction multi-utilisateur et distante. Les méthodologies utilisant des connaissances *a priori*, quant à elles, proposent un processus supervisé de construction semi-automatique d'ontologies. Elles se distinguent selon les types de données en entrée : textes, dictionnaires, bases de connaissances, schémas de base de données, etc.. Dans ce chapitre, nous nous intéressons particulièrement à la construction à partir de textes où des ressources linguistiques brutes sont en général automatiquement filtrées (fouille de textes) par des outils de Traitement Automatique de la Langue (TAL), puis utilisées pour construire des concepts d'une ontologie. Ce chapitre présente donc tour à tour les notions de terminologie (Cf. section 1.2), de thésaurus (Cf. section 1.3) et d'ontologie (Cf. section 1.4). Nous présentons également, dans la section 1.5, quelques approches de construction d'ontologies ainsi que les outils associés à ces approches.

1.2 Notion de Terminologie

En ingénierie de la connaissance, les terminologies sont utilisées pour représenter, de façon informelle, un domaine d'étude. Dans cette section, nous présentons la notion de terminologie ainsi que les éléments la constituant.

1.2.1 Définition

Une terminologie présente l'ensemble des termes particuliers à une science, un domaine ou un art pour un groupe de personnes (Office de la langue française, 2001). Philippe Lefèvre, dans son livre sur la recherche d'informations [Lefèvre, 2000], propose une définition plus précise : *"les terminologies sont des listes de termes d'un domaine ou sujet donné représentant les concepts ou notions les plus fréquemment utilisés ou les plus caractéristiques, cette liste étant ou non structurée"*. La terminologie consiste également en l'étude du choix et de l'usage des termes faisant partie des vocabulaires de spécialité, qu'on peut trouver dans plusieurs domaines de connaissance. Par exemple, le terme *table* relève à la fois de la terminologie de *l'ameublement* et de celle de *l'informatique*.

La terminologie, considérée comme science, s'intéresse au recensement des termes désignant les concepts d'un domaine pour faciliter l'échange de connaissances dans une langue et d'une langue à une autre. Pour y parvenir on a en général besoin :

- d'identifier les concepts ;
- de normaliser et figer l'expression des concepts du domaine en fixant les termes qui les désignent ;

– de rendre compte de l'agencement relatif des concepts recensés en étudiant les relations linguistiques existant entre les termes désignant les concepts [Wüster, 1981].

La conception terminologique du monde s'appuie donc sur trois notions principales :

1. *la chose*, c'est-à-dire l'objet du monde, aussi qualifié de *référent* ;
2. *le signe*, aussi qualifié de *forme* ou de *signifiant*, c'est-à-dire le terme, la chaîne de caractères, ou la photo qui symbolise cet objet particulier du monde ;
3. *le concept*, encore appelé *signifié*, correspond à l'idée que dénote le *signe*.

Cette catégorisation du monde, connue sous le nom de triangle sémiotique (Cf. figure 1.1) et proposée en 1923 par Charles Ogden et Ivor Richards [Charles Ogden, 1923], permet de faire la différence entre un objet du monde réel, sa représentation et son interprétation dans le monde.

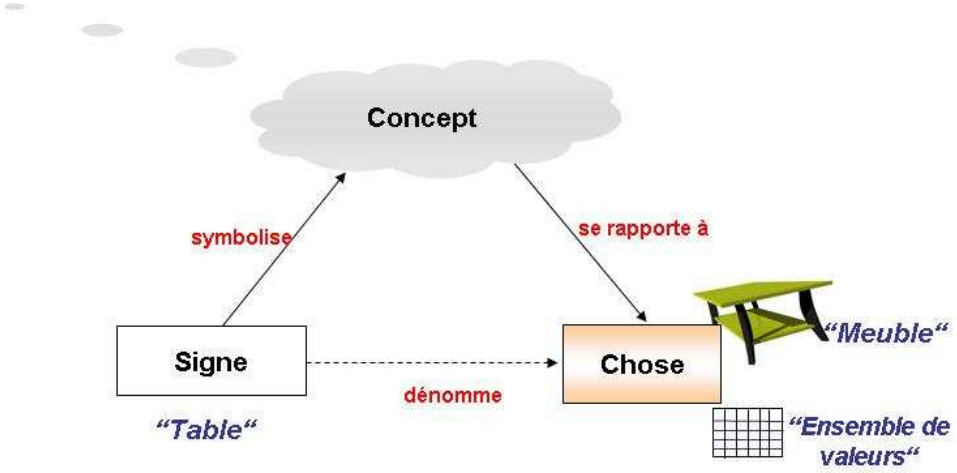

FIGURE 1.1 – Triangle Sémiotique.

La notion de triangle sémiotique est à la base de l'hypothèse selon laquelle il est possible de rattacher des mots d'un corpus à des concepts du monde réel représentés au sein d'une ontologie.

1.2.2 Constituants d'une terminologie

Les termes présents dans la terminologie peuvent être reliés par des relations qui témoignent, par exemple, d'un rapport de spécialisation-généralisation (relation d'hyponymie par exemple). On dit par exemple qu'une *table* est plus spécifique qu'un *meuble* (*table* est hyponyme de *meuble*). Ainsi, une terminologie représente des concepts sous la forme d'un système de termes normalisés.

La principale insuffisance d'une terminologie réside dans son niveau de structuration assez limité. En effet, les terminologies ne représentent que les relations entre termes et se limitent en général à des relations purement linguistiques (synonymie, hyponymie, etc.). L'enrichissement du niveau descriptif des terminologies a conduit à la notion de thésaurus que nous présentons dans la section suivante.

1.3 Notion de Thésaurus

Afin de se rapprocher davantage de la représentation des relations entre les objets du monde réel, les thésaurus ont rajouté aux terminologies un ensemble de relations dites sémantiques (*relation de préférence* par exemple). Dans cette section, nous présentons brièvement la notion de thésaurus.

1.3.1 Définition

Selon Foskett [Foskett, 1997], un thésaurus est un ensemble structuré de termes d'un vocabulaire ; par exemple les termes techniques utilisés dans un domaine précis, représentés de façon normalisée par des descripteurs ou des mots clés. Un thésaurus est donc un outil linguistique qui permet de traduire, en un langage artificiel dépourvu d'ambiguïté, des mots exprimés en langage naturel. mettre en relation le langage naturel des utilisateurs et celui contenu dans les ressources documentaires. Cette technique pallie les limites du langage naturel, très riche mais souvent ambigü. Le thésaurus évite ainsi les risques induits par les synonymies, les homonymies et les polysémies constatées dans le langage naturel. Contrairement à un dictionnaire auquel il est souvent assimilé, un thésaurus ne fournit qu'accessoirement des définitions ; les relations entre les termes et leur sélection l'emportant sur la description des significations.

1.3.2 Constituants d'un Thésaurus

Un thésaurus est constitué d'un ensemble organisé de termes, choisis pour leur capacité à faciliter la description d'un domaine et à harmoniser la communication et le traitement de l'information. Les termes d'un thésaurus sont reliés entre eux par des relations sémantiques (hiérarchique, équivalence, etc.). Ainsi, pour des besoins particulier propre à un domaine d'étude, un terme, encore appelé *descripteur*, est aussi peu ambigu que possible et peut être préféré à des termes voisins ou *non-descripteurs* ; traduisant de ce fait l'existence d'une relation d'équivalence entre les termes considérés. En pratique, un thésaurus fournit un répertoire structuré de termes pour l'analyse du contenu, le classement et donc l'indexation de documents. Plusieurs langage de modélisation permettent de représenter les thésaurus. En guise d'illustration, nous utilisons le langage SKOS [4]. Ainsi, dans la figure 1.2 qui présente un sous-ensemble du thésaurus modélisant les personnes d'une université, les *descripteurs* sont représentés par la propriété `skos:prefLabel` tandis que les *non-descripteurs* sont représentés par la propriété `skos:altLabel`.

Bien que la structuration et l'ajout de relations sémantiques permettent de mieux décrire les relations entre les objets du monde réel, le pouvoir d'expression de ces relations reste assez limité. En effet, un thésaurus est difficilement utilisable représenter les relations propres à un domaine comme exprimer le fait qu'entre un `"Enseignant"` et un `"Cours"` il existe la relation `"est dispensé par"`. Dans un thésaurus, on spécifie juste que `"Enseignant"` et `"Cours"` sont des entités reliées (en utilisant la notion de `skos:related` par exemple).

En introduisant des notions telles que les *classes* et les *propriétés*, les ontologies disposent d'un pouvoir d'expression plus riche.

4. Simple Knowledge Organisation System:http://www.w3.org/2004/02/skos/

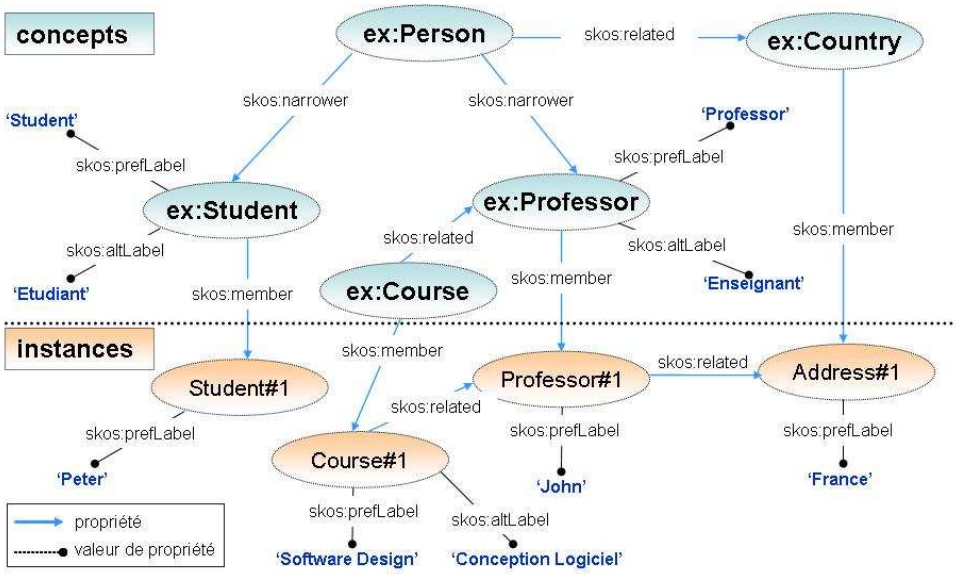

FIGURE 1.2 – Exemple d'un thésaurus en SKOS.

1.4 Notion d'ontologie

Aussi bien dans les terminologies que dans les thésaurus, le niveau de description ne concerne que les termes. Les terminologies et les thésaurus sont donc peu adaptés pour représenter des relations entre concepts. Les ontologies, par contre, visent à expliciter la définition des concepts et les relations entre ces concepts.

1.4.1 Définition

Plusieurs définitions ont été proposées pour le terme ontologie, celles-ci dépendant en général du domaine d'application. Dans cette section nous présentons quelques unes de ces définitions.

1.4.1.1 En philosophie

En philosophie, le terme ontologie désigne le domaine d'étude qui s'intéresse à *"l'étude de l'être en tant qu'être, indépendamment de ses déterminations particulières"* [5] ; laissant ainsi sous-entendre un discours indépendant d'un quelconque point de vue. Il importe donc de faire une distinction entre *"l'observateur"* des choses et les choses *"observées"*. En philosophie, il existe trois hypothèses sur l'appréhension de la connaissance [Dietz, 2006] :

– *l'objectivisme* défend l'hypothèse selon laquelle le monde existe par lui-même, de façon indépendante de *l'observateur*. L'observateur croit en la vérité d'une réalité objective ;

– *le subjectivisme* défend l'hypothèse selon laquelle la réalité n'existe pas en dehors du sujet qui

5. Dictionnaire Antidote

l'observe et qu'à la limite, chaque observateur a sa propre image de la réalité ;

– *le constructivisme* admet l'hypothèse selon laquelle il n'y a pas d'objectivité absolue de la réalité, que la réalité existe aussi par l'interprétation de celle-ci. En contrepartie, il admet aussi que les éléments interprétés font partie d'une réalité que l'on pourrait qualifier de semi-objective.

Pour Dietz [Dietz, 2006], l'ontologie s'interprète selon un point de vue constructiviste. Elle est une composante particulière de la réalité qui sert de base de communication au discours sur une partie de la réalité (objectivisme).

1.4.1.2 En informatique

Dans le domaine de l'informatique, la définition du terme ontologie la plus citée dans la littérature est celle proposée par Gruber [Gruber, 1993] : *"An ontology is an explicit specification of a conceptualization"*. [Traduction: Une ontologie est une spécification explicite d'une conceptualisation]. D'autres définitions pertinentes de la notion d'ontologie pour l'ingénierie ontologique ont été proposées. Ces définitions sont souvent des raffinements de définitions déjà proposées et/ou sont complémentaires avec elles. [Studer et al., 1998] par exemple, dira d'une ontologie qu'elle est une spécification formelle, explicite d'une conceptualisation partagée. Selon [Pierra et al., 2005], *"une ontologie est une collection de descriptions explicites, formelles et consensuelles de l'ensemble des concepts d'un domaine dans le contexte le plus large où ces concepts ont un sens précis, sans aucune restriction ou règle correspondant à une utilisation particulière"*.

La notion de *description explicite* signifie que, face à un objet matériel ou un artefact donné appartenant au domaine ciblé par une ontologie, un utilisateur humain de l'ontologie doit savoir décider :
– à quelle(s) catégorie(s) l'objet appartient et n'appartient pas ;
– quelle(s) propriétés s'appliquent à l'objet ;
– quelle(s) grandeur(s) ou valeur(s) caractéristique(s) corresponde(nt) à chaque propriété applicable.

La notion de *description formelle* signifie que l'ontologie doit définir tous les axiomes et relations nécessaires et vérifiables par un agent informatique.

Bien que ces définitions soient communément admises, la représentation des composantes d'une ontologie dépend du modèle d'ontologies utilisé. La section suivante présente les constituants de base d'une ontologie.

1.4.2 Constituants d'une ontologie

En informatique, deux types d'ontologies sont fréquemment distingués. Les *ontologies de haut niveau* fournissent des définitions pour des concepts généraux tels que les notions de processus, d'objet ou d'événement afin de servir de fondement pour des ontologies plus spécifiques dites de domaine [Ian Niles, 2001, Gangemi et al., 2003]. Les *ontologies de domaine* sont liées à un univers du discours particulier. Elles décrivent et représentent la connaissance existant dans le domaine correspondant à cet univers. Les ontologies de domaine sont plus spécifiques que les ontologies noyaux de domaine (appelées

core-ontologies) qui englobent les concepts généraux d'une discipline, par exemple le droit.

Dans ce travail de thèse, nous ne nous intéressons qu'aux ontologies de domaine qui permettent de décrire la sémantique des objets d'un domaine d'étude. Ainsi, dans ce mémoire, le mot ontologie est utilisé pour désigner une ontologie de domaine.

Ces ontologies de domaine représentent la sémantique des concepts d'un domaine en termes de *classes* et de *propriétés*. Une classe, aussi appelée *concept*, regroupe et abstrait les objets du domaine possédant des caractéristiques communes. Une propriété, aussi appelée *attribut* ou *rôle*, permet de caractériser les objets du domaine par une ou plusieurs valeurs. Elle peut être définie sur un *domaine* indiquant la classe d'objets qu'elle permet de décrire et associée à un *codomaine* indiquant le type de données dans lequel elle peut prendre ses valeurs. Les classes ont une *extension* constituée d'un ensemble d'*instances* aussi appelées *individus*, et qui désignent des objets du domaine. Une instance est décrite par son appartenance à une ou plusieurs classes et par un ensemble de valeurs de propriétés.

Pour définir des ontologies, un *modèle d'ontologies* permettant de représenter ces ontologies est nécessaire. Dans la littérature, il existe plusieurs modèles d'ontologies tels que RDFS [Manola and Miller, 2004] et OWL [Dean and Schreiber, 2004, Smith et al., 2004, Patel-Schneider et al., 2004] utilisés dans le web sémantique, PLIB [ISO13584-42, 1998, ISO13584-25, 2004] utilisé en ingénierie, etc..

Définition 1.4.1

Formellement [Pierra et al., 2005] définit une ontologie par un quadruplet : $O = \langle C, \mathcal{P}, Sub, Applic \rangle$, avec :

- *C : ensemble des classes utilisées pour décrire les concepts d'un domaine donné (comme les services de voyages, les pannes des équipements, les composants électroniques, etc.). Chaque classe est associée à un identifiant universel globalement unique (GUI) ;*
- *$\mathcal{P}$: ensemble des propriétés susceptibles d'être utilisées pour décrire les instances de l'ensemble des classes C. Une propriété peut aussi bien avoir comme co-domaine un type (on la qualifie alors aussi d'attribut) qu'une classe, ou un ensemble de classes (on la qualifie aussi de relation). Chaque propriété est associée à un GUI ;*
- *$Sub : C \rightarrow 2^C$ est la relation de subsomption qui, à chaque classe c_i de l'ontologie, associe ses classes subsumées directes, c'est-à-dire les classes vérifiant la propriété $\forall c_1, c_2 \in C$, c_1 subsume c_2 si et seulement si $\forall x \in c_2, x \in c_1$. Sub définit un ordre partiel sur C et 2^C désigne l'ensemble de partie de C ;*
- *$Applic : C \rightarrow 2^{\mathcal{P}}$ associe à chaque classe de l'ontologie les propriétés qui sont applicables pour chaque instance de cette classe.*

La figure 1.3 présente un sous-ensemble d'une ontologie modélisant les personnes d'une université. Dans cette ontologie, une personne (`Person`), identifiée par son nom (`name`) , son age (`age`) et son pays d'origine (`Country`), est soit un étudiant (`Student`) soit un enseignant (`Professor`). De plus, un enseignant dispense des cours (`Course`) aux étudiants.

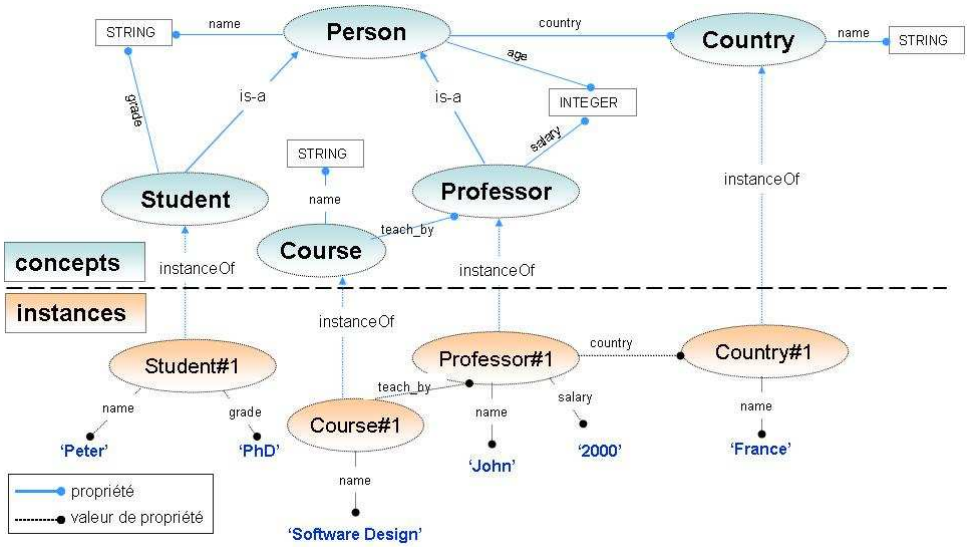

FIGURE 1.3 – Exemple d'une ontologie et de ses instances.

Maintenant que nous savons ce qu'est une ontologie et comment elle pourrait être représentée, nous présentons, dans la section suivante, quelques domaines d'utilisation des ontologies.

1.4.3 Intérêt des ontologies

De nos jours, plusieurs applications informatiques utilisent les ontologies comme modèle d'un domaine. Cette utilisation des ontologies permet notamment de résoudre différents problèmes tels que la conception et l'indexation des bases de données, l'intégration et le partage de données, le web sémantique, etc.. Dans cette section, nous présentons l'intérêt de l'utilisation des ontologies dans ces différents domaines d'application.

1.4.3.1 Conception/Indexation de bases de données

Une ontologie étant une conceptualisation d'un domaine d'étude, elle peut être utilisée comme connaissance *a priori* pour la conception d'une base de données. Cette approche est suivie dans [del Mar Roldán García et al., 2005], [Sugumaran and Storey, 2006] et [Dehainsala et al., 2007b]. De manière générale, la démarche proposée dans ces approches se résume de la manière suivante :

1. choisir une ontologie couvrant le domaine d'étude sur lequel porte l'application pour laquelle la base de données est conçue ;
2. étendre (si besoin y est) cette ontologie pour ajouter les concepts nécessaires mais qui ne seraient pas représentés ;
3. choisir le sous-ensemble de cette ontologie qui couvre les besoins de l'application pour laquelle la base de données est conçue ;
4. implanter ce sous-ensemble dans la base de données.

Une ontologie peut également être utilisée pour enrichir la sémantique du modèle logique d'une base de données en l'annotant. Cette *indexation sémantique* des bases de données consiste à associer les divers éléments d'un modèle logique (tables, colonnes, contraintes, etc.) à une ontologie. La définition de ces correspondances nécessite un langage de correspondance (*mapping*) tel que R_2O [Barrasa et al., 2004]. Cette définition de correspondance peut être assistée par des outils de matching qui proposent des correspondances en s'appuyant sur les noms utilisés [del Mar Roldán García et al., 2005, Sugumaran and Storey, 2006] ou sur la structure du modèle logique traité comme par exemple, les clés étrangères [An et al., 2006].

1.4.3.2 Intégration de données

Un système d'intégration fournit une interface d'accès unique à des données stockées dans plusieurs sources de données. Ces sources de données sont en général conçues indépendamment l'une de l'autre par des concepteurs différents. Par conséquent, des données relatives à un même domaine d'étude peuvent être représentées différemment dans ces différentes sources. C'est le problème de l'hétérogénéité des données. Goh [Goh, 1997] a identifié trois principales causes à l'hétérogénéité sémantique des données :

- *les conflits de nom* ont lieu lorsque des noms différents sont utilisés pour décrire le même concept (synonyme) ou lorsque le même nom est utilisé pour des concepts différents (homonyme) ;
- *les conflits de mesure de valeur* ont lieu lorsque différents systèmes de référence sont utilisés pour évaluer une valeur. C'est le cas par exemple, lorsque des unités de mesure différents sont utilisées par les sources de données ;
- *les conflits de contexte* ont lieu lorsque des concepts semblent avoir la même signification mais diffèrent en réalité en raison des différents contextes de définition ou d'évaluation.

Étant donné qu'une ontologie peut servir de pivot pour définir la sémantique des données des différentes sources à l'aide de concepts communs, formalisés et référençables, leur utilisation est une solution pour résoudre les problèmes d'hétérogénéité des données. Différentes propositions d'intégration basées sur des ontologies ont ainsi été faites. Un état de l'art de ces approches est présenté dans [Wache et al., 2001, Noy, 2004].

1.4.3.3 Partage de données

Une conceptualisation consensuelle d'un domaine dans lequel chaque élément est référençable, peut être utilisée comme un format d'échange de données sur ce domaine [Chawathe et al., 1994, ISO13584-42, 1998]. Contrairement au format d'échange usuel qui spécifie la structure complète des données échangées et où la signification de chaque élément résulte de sa position dans la structure globale, les échanges basés sur des ontologies peuvent être très flexibles. En effet, dans ce type d'échange, la signification de chaque élément peut être définie localement en référençant des identifiants d'éléments d'une ontologie. Cette capacité fait que des structures d'échanges différentes peuvent être interprétées de façon non ambigüe par un même système receveur.

1.4.3.4 Le Web

La généralisation des accès haut débit au réseau Internet a provoqué un fort engouement pour ce média de communication. Autrefois réservé aux professionnels de l'informatique avec un contenu essentiellement scientifique, celui-ci est aujourd'hui accessible à tous pour des utilisations variées. Que ce soit pour faire ses démarches administratives, faire des achats en ligne, rechercher un emploi, etc., tout le monde a perçu les bénéfices de ce vecteur de communication.

Cette excellente source d'information souffre néanmoins d'un défaut majeur capable de décourager des utilisateurs débutants. Alors que l'on parle d'une toile pour décrire ce réseau, l'ensemble des services qui y sont offerts sont parfois assez isolés. Par conséquent, pour arriver au résultat escompté, un utilisateur doit, soit avoir une connaissance approfondie du Web, soit passer par une période fastidieuse de navigation sur plusieurs sites web. De plus, l'évolution très rapide et incontrôlée des services offerts sur le Web conduit souvent à la seconde situation, même pour des utilisateurs expérimentés. Cette difficulté vient du fait que les noms utilisés pour décrire un même service sont très différents et dépendent du créateur du service. L'idée du Web Sémantique [Berners-Lee et al., 2001] est de développer des ontologies de domaine puis d'indexer les services par leur description en termes de ces ontologies, rendant ainsi la recherche automatisable et réalisable par des agents informatiques.

1.4.3.5 Interopérabilité des logiciels

L'utilisation d'un modèle comme une spécification d'un logiciel est à la base de l'approche MDA (Model-Driven Archictecture). Dans cette approche, un modèle est utilisé pour générer le code de l'application. Le lien formel entre le modèle et le code permet alors de faire évoluer ce dernier lorsque la spécification du logiciel évolue. Actuellement, plusieurs logiciels traitant des problèmes similaires sur le même domaine sont généralement définis en utilisant différents modèles. Par conséquent, on n'est pas à l'abri d'un problème d'interopérabilité entre ces logiciels. L'utilisation d'ontologies peut contribuer à résoudre ce problème. En effet, puisque les ontologies sont consensuelles, les différents modèles d'un même domaine définis pour plusieurs logiciels peuvent être appariés à une ontologie de domaine. Ces logiciels peuvent alors interagir en utilisant les accesseurs fournis par cette ontologie. Cette approche est connue sous le nom d'*ingénierie logicielle dirigée par les ontologies* [Tetlow et al., 2005].

1.4.3.6 Traitement du langage naturel

Le traitement du langage naturel aborde entre autres le problème de la compréhension du langage humain par un ordinateur. L'analyse syntaxique et sémantique du langage naturel est une étape clé pour la résolution de ce problème. Les ontologies peuvent être utilisées dans ces étapes pour, d'une part, construire le lexique utilisé lors de l'analyse syntaxique d'un texte, et, d'autre part, pour effectuer des traitements complexes lors de l'analyse sémantique du texte tel que la résolution des problèmes de polysémie. Cette approche est par exemple suivie dans [Estival et al., 2004].

Les techniques développées dans le cadre du traitement du langage naturel sont notamment utilisées pour la recherche d'information. En effet, la détection de synonymie entre les termes permet d'améliorer la recherche documentaire. Dans les moteurs de recherche, une requête est en général composée d'un

ensemble de termes éventuellement connectés par les opérateurs logiques OU, ET et NON. Le moteur de recherche produit essentiellement sa réponse en fonction des termes contenus dans les documents parcourus. De nombreuses propositions (Cf. [Haav and Lubi, 2001] pour un état de l'art) préconisent l'utilisation des ontologies dans les moteurs de recherche pour permettre de retourner également les documents pertinents par rapport à la signification des termes contenus dans la requête. En utilisant des ontologies, les moteurs de recherche n'effectuent plus seulement une recherche par mots clés mais aussi par concepts structurés au sein d'une ontologie car un mot d'un document est désormais rattaché à un concept d'une ontologie.

Maintenant que nous savons ce que sont les ontologies et à quoi elles pourraient eventuellement servir, nous allons nous intéresser à la manière de les construire.

1.5 Construction d'ontologies : méthodes et outils

Plusieurs outils permettent de construire des ontologies. Dans cette section, nous proposons de regrouper ces outils en deux principaux groupes. D'une part, nous avons des éditeurs qui supposent que l'ontologie est déjà conçue (sur papier par exemple), et donc qu'il ne reste plus qu'à l'éditer pour la rendre exploitable par un agent informatique ; et, d'autre part, nous avons des outils d'aide à la construction. Ces derniers permettent une conception supervisée (semi-automatique) comportant en général plusieurs étapes pour la construction d'une ontologie. Dans cette section, nous discutons des caractéristiques de quelques méthodes et outils de construction d'ontologies.

1.5.1 Les éditeurs

Dans cette section, nous illustrons le rôle des éditeurs d'ontologies au travers des éditeurs tels que Protégé [6], PLibEditor [7], OntoEdit [Sure et al., 2002] et DOE [8]. Outre leurs interfaces conviviales, ces outils ont en commun le fait qu'ils supposent que l'ontologie existe déjà, et, par conséquent, il ne reste plus qu'à l'éditer. Le pouvoir expressif informel de DOE, l'architecture extensible de Protégé ainsi que l'architecture de base de données sur laquelle s'appuie PLibEditor constituent des caractéristiques complémentaires qui permettent de donner une vue globale de l'ensemble des éditeurs existants.

1.5.1.1 Protégé

Développé à l'Université de Stanford, Protégé [Gennari et al., 2003] est actuellement l'environnement d'édition d'ontologies le plus utilisé. Initialement basé sur le modèle des frames [Minsky, 1975], la version actuelle de Protégé permet la construction d'ontologies conformes au modèle d'ontologies OWL (Cf. figure 1.4). Cet éditeur offre tous les artefacts nécessaires pour l'édition des différents éléments d'une ontologie OWL (concepts, propriétés, instances), avec la possibilité de spécifier des contraintes et d'utiliser des moteurs d'inférence externes tels que Racer [Volker Haarslev, 2001] ou Pellet [Sirin et al., 2007] pour vérifier la consistance de l'ontologie et d'inférer de nouvelles connaissances.

6. http://www.protege.stanford.edu
7. http:// www.plib.ensma.fr
8. Differential Ontology Editor http://homepages.cwi.nl/ troncy/DOE/

Protégé s'enrichit régulièrement de l'apport de la communauté des utilisateurs et développeurs grâce à son architecture à base plug-ins qui permet d'étendre ses fonctionnalités. L'outil permet désormais d'intégrer plusieurs ontologies et de gérer les différentes versions d'une même ontologie.

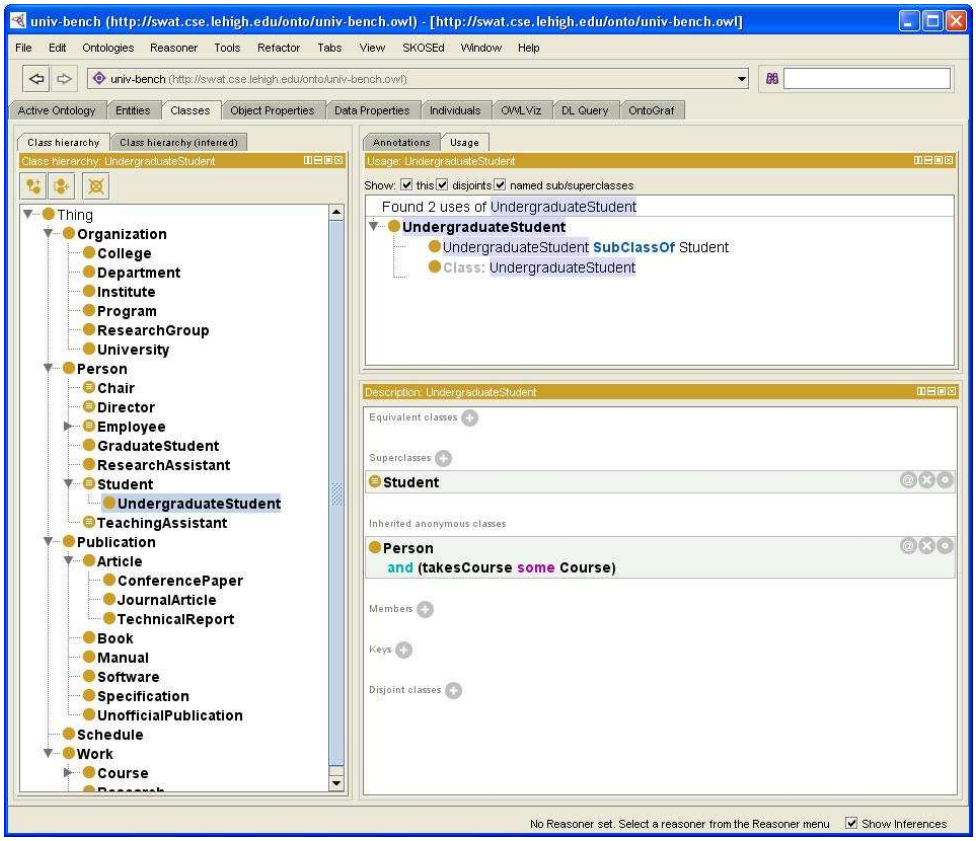

FIGURE 1.4 – Protégé.

1.5.1.2 PLibEditor

PLibEditor [9] (Cf. figure 1.5) permet de créer des ontologies conformes au modèle d'ontologies PLIB. Cet éditeur s'appuie sur une architecture de base de données dite à base ontologique (OntoDB [Dehainsala et al., 2007b]) qui permet de stocker dans la base de données, à la fois le modèle d'ontologies, l'ontologie et les instances. PLIBEditor, a la particularité de manipuler également les ontologies conformes au modèle d'ontologies OWL. En effet, dans sa deuxième version [Fankam et al., 2008], OntoDB permet de stocker à la fois les modèles d'ontologies PLIB et OWL et assure une transformation entre ces deux modèles. PLIBEditor permet également de vérifier le contenu d'une ontologie ou d'un ensemble d'instances à base ontologique. Il est aussi possible de raisonner grâce à certains constructeurs d'ontologies. Enfin, PLIBEditor peut importer et exporter des ontologies et leurs instances dans le format d'échange normalisé PLIB.

9. http:// www.plib.ensma.fr

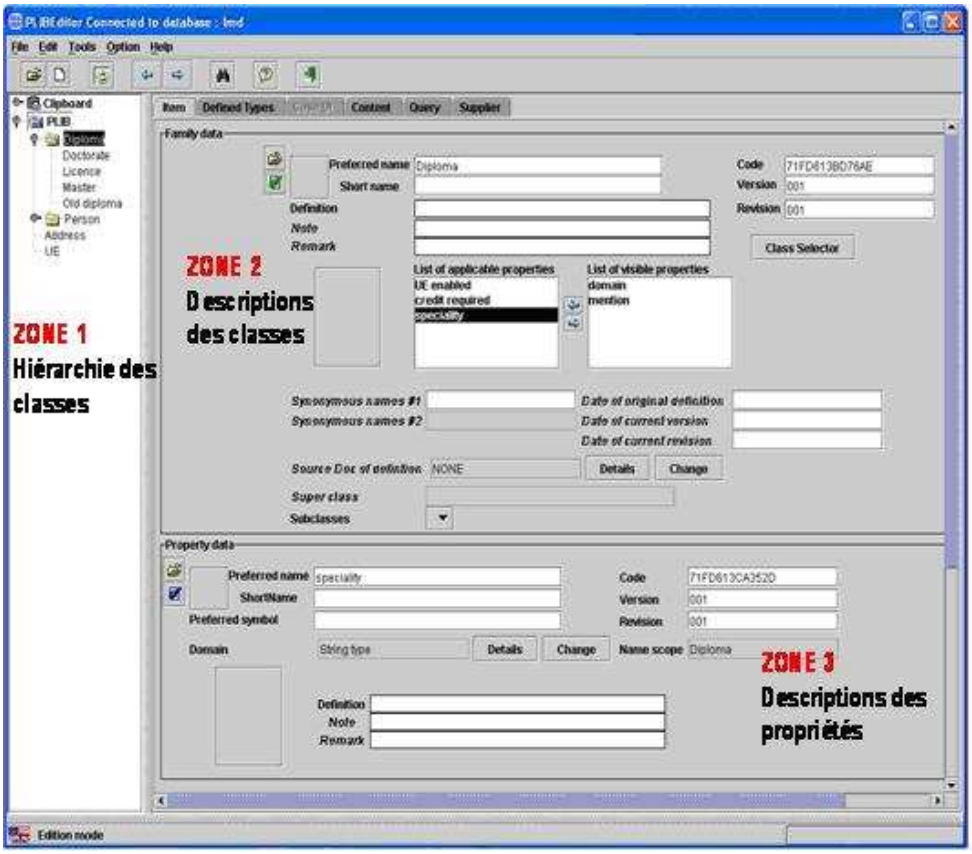

FIGURE 1.5 – PLibEditor.

1.5.1.3 OntoEdit

OntoEdit [Sure et al., 2002, Sure et al., 2003] est un outil de construction d'ontologies développé par le Knowledge Management Group de l'Université de Karlsruhe. Il fournit (Cf. 1.6) un environnement graphique d'édition d'ontologies permettant l'inspection, la navigation, le codage et la modification d'une ontologie. De nouvelles fonctionnalités peuvent être ajoutées à l'outil grâce à de nouveaux plug-ins. OntoEdit fournit des sous-ensembles fonctionnels permettant d'exporter une ontologie conformes à plusieurs langages de représentation (XML, Flogic, RDF(S), DAML+OIL). Dans sa version commerciale (permettant le stockage des ontologies dans une base de données relationnelle), OntoEdit fait partie de la suite logicielle proposée par la société Ontoprise.

1.5.1.4 DOE

Contrairement à Protégé, OntoEdit et PLibEditor qui s'intéresse davantage à la représentation formelle des concepts d'une ontologie, l'éditeur DOE [Isaac and Malaisé, 2003] (Cf. figure 1.7) préconise la structuration de la description informelle pour décrire plus précisément les concepts. Cet éditeur utilise pour cela une sémantique dite "différentielle" pour documenter les hiérarchies de généralisation/spécialisation en appliquant quatre règles [Bachimont, 2000] :

– le principe de *communauté avec le père* selon lequel il faut expliciter en quoi le fils est identique

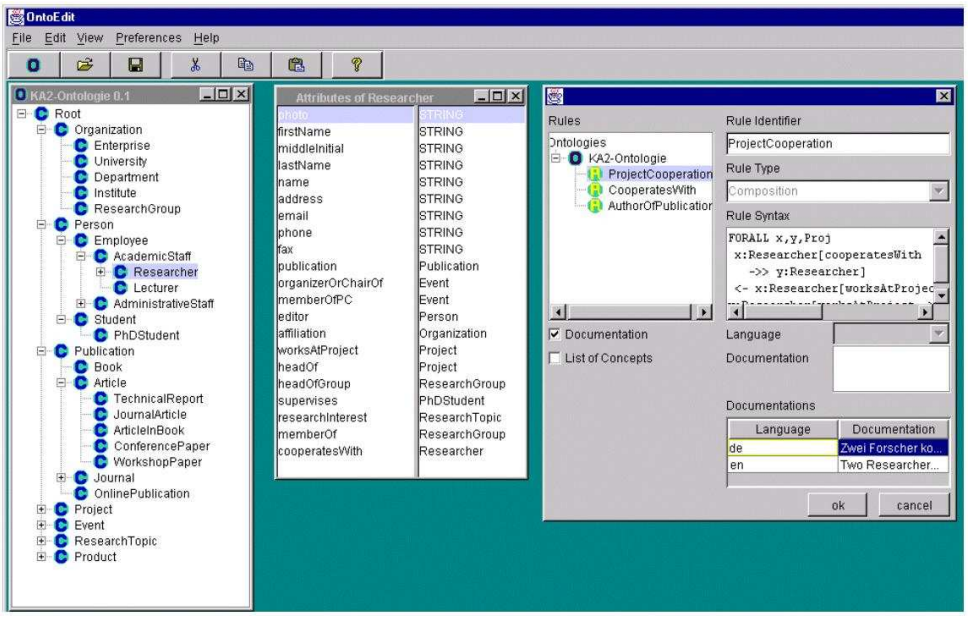

FIGURE 1.6 – OntoEdit.

au père qui le subsume ;

– le principe de *différence avec le père* selon lequel il faut expliciter en quoi le fils est différent du père qui le subsume ;

– le principe de différence avec les frères selon lequel il faut expliciter la différence de la notion considérée avec chacune des notions représentées par les classes de même père ;

– le principe de *communauté avec les frères* selon lequel il faut expliciter la communauté existante entre la notion considérée et chacune des notions représentées par les classes de même père.

L'application de ces principes est à la base de la notion d'ontologie différentielle qui se traduit dans DOE par la présence d'informations textuelles renseignées lors de la création d'un nouveau concept dans une hiérarchie.

Nous pouvons compléter cette liste non exhaustive d'éditeurs par des outils tels que SWOOP [10], KMgen [11], TopBraid Composer [12], KAD-Office [13], etc..

Si ces éditeurs complémentaires constituent désormais des outils de référence en ingénierie ontologique, ils ne sont utilisables que si l'ontologie est déjà connue, c'est-à-dire si elle a déjà été conçue sur papier. Or, cette conception, en raison de son caractère consensuel, est parfois fastidieuse. Certains

10. Semantic Web Ontology Editor: http://code.google.com/p/swoop/
11. http://www.algo.be/dev-logiciels.htm#kmed
12. http://www.topquadrant.com/products/TB_Composer.html
13. http://www.iknova.com/produits.htm

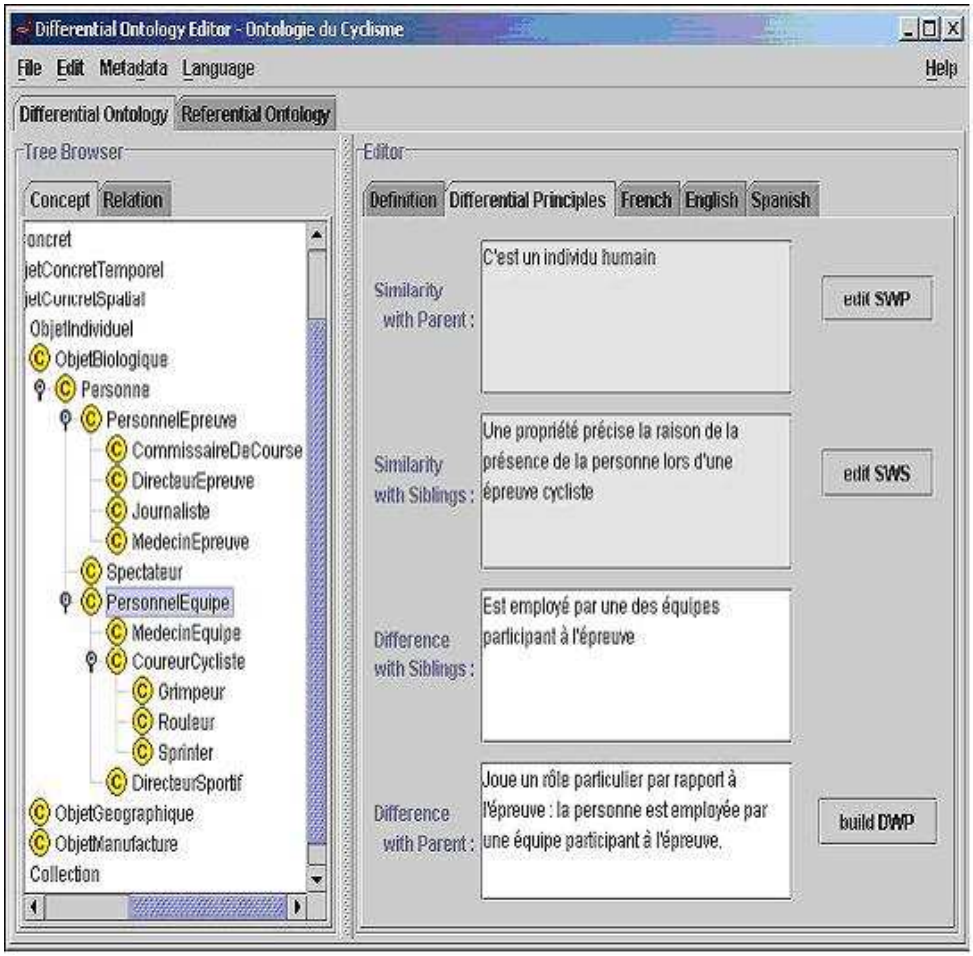

FIGURE 1.7 – DOE.

outils, par contre, proposent des approches de construction permettant d'alléger cette difficulté : on parle d'outils d'aide à la construction. L'un des scénarios de construction le plus répandu est la construction d'ontologies à partir de textes.

1.5.2 Les outils d'aide à la construction : construction à partir de textes

La construction d'ontologies à partir de textes s'intéresse à la possibilité d'utiliser des corpus de textes, en extraire des *termes* et des *relations* entre *termes* pour ensuite construire une ontologie à partir de ces *termes* et *relations*. Dans cette section, nous présentons quelques outils de construction d'ontologies à partir de textes.

1.5.2.1 Intérêt de l'utilisation de textes

Plusieurs raisons motivent l'utilisation des textes pour la construction d'ontologies. Parmi ces raisons, nous pouvons citer :

– l'indexation documentaire et la recherche d'information ;

– la préservation du lien entre l'ontologie et les éléments textuels justifiant la modélisation choisie ;
– la proposition, pour une communauté, des modèles dont les noms des concepts reflètent au mieux le vocabulaire utilisé par les usagers de la communauté ;
– la nécessité de rendre compte, au plus juste, de la connaissance telle qu'elle est explicitée dans des documents consensuels et partagés ;
– l'accélération de la construction des modèles grâce à l'automatisation de l'analyse (utilisation de logiciels de TAL) ;
– etc.

Si plusieurs démarches peuvent être utilisées pour construire une ontologie à partir de textes, [Paul Buitelaar, 2005] a identifié un ensemble de tâches communes participant au processus incrémental de construction d'ontologies dans le contexte de l'extraction semi-automatique de connaissances à partir de corpus de textes non structurés (Cf. figure 1.8). Parmi ces tâches, nous pouvons citer l'identification des :

– termes pertinents pour l'ontologie à construire ;
– relations de synonymie entre les termes ;
– concepts et de leurs attributs ;
– relations taxonomiques (structure de la hiérarchie) entre concepts ;
– relations non taxonomiques (génériques) entre concepts ;
– règles spécifiant les contraintes sur les propriétés des concepts et les relations entre concepts.

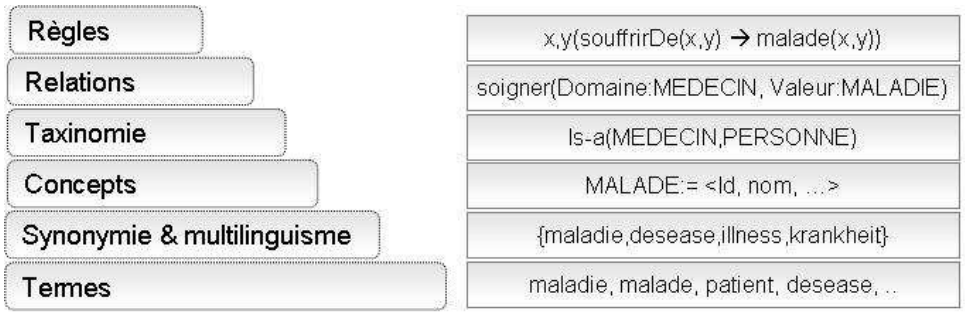

FIGURE 1.8 – Tâches liées à la construction d'ontologies.

[Biebow and Szulman, 1999], pour leur part, proposent une démarche plus générale consistant en :

– *constitution d'un corpus* (documents techniques, comptes rendus, livres de cours, etc.), à partir d'une analyse des besoins de l'application visée ;
– *étude linguistique*, pour identifier des termes et des relations lexicales, en utilisant des outils de traitement de la langue naturelle ;
– *normalisation sémantique*, conduisant à des concepts et des relations sémantiques définis dans un langage semi-formel ;
– *formalisation* et *intégration* des concepts au sein d'une base de connaissance formelle.

Toutefois, quel que soit la démarche utilisée, la construction d'ontologies à partir de corpus de textes requiert une analyse linguistique des textes.

1.5.2.2 Analyse linguistique

Dans le processus de construction d'ontologies à partir de textes, la notion de *candidats termes* est utilisée pour désigner les mots ou groupe de mots (syntagmes) extraits par les outils de TAL. Ces *candidats termes* sont ensuite validés par l'utilisateur et acquièrent dès lors le statut de *termes*. En général, pour valider les *candidats termes*, l'utilisateur procède soit à une analyse statistique, soit à une analyse distributionnelle.

Analyse statistique d'un corpus

L'analyse statistique d'un *terme/candidat terme* consiste, en général, à étudier des paramètres renseignant en général sur la fréquence d'apparition de ce *terme/candidat terme* dans le corpus. Dans cette section nous décrivons quelques paramètres d'analyse statistique et leur utilisation.

1. **Fréquence de Terme.**
   La fréquence de terme est proportionnelle au nombre de fois qu'un terme apparaît dans un document. Ce nombre est généralement normalisé par le nombre total de *termes* dans un document afin d'éviter les biais pour de longs documents. La fréquence de terme est parfois utilisée pour :
   – identifier les termes pertinents d'un corpus ;
   – identifier les candidats de concepts et de relations, c'est-à-dire distinguer parmi les *termes*, ceux qui deviendront des classes ou des propriétés de l'ontologie ;
   – etc.

2. **Fréquence Inverse de Document.**
   Le TF-IDF (*Term Frequency - Inverse Document Frequency*) [Salton et al., 1982] est une mesure statistique utilisée pour évaluer l'importance (le poids) d'un mot dans un document d'un corpus. L'importance croit proportionnellement avec le nombre d'occurrences du mot dans le document mais est contrebalancée par la fréquence du mot dans le corpus. La fréquence inverse du document (IDF) pour un terme donné est une mesure de l'importance générale de ce terme.
   Pour un terme t_i, on a :

   • $\mathrm{IDF}(t_i) = \log(\frac{\|D\|}{\|\{d_j \mid t_i \in d_j\}\|})$ où :

   – $\|D\|$: nombre total de documents dans le corpus ;

   – $\left\|\{d_j \mid t_i \in d_j\}\right\|$: nombre de documents où le terme t_i apparaît.

   • $(\mathrm{TF}-\mathrm{IDF})(t_i) = \mathrm{TF}(t_i) \times \mathrm{IDF}(t_i)$.

   Ces formules montrent qu'un haut poids TF-IDF est atteint avec une haute fréquence d'un terme dans un document donné et une faible occurrence de ce terme dans les documents du corpus. Le TF-IDF tend à filtrer les termes communs et est parfois utilisé comme métrique de similarité pour mesurer la distance entre des termes. Cette métrique permet ainsi, d'une part, de regrouper les

termes similaires en concepts communs, et, d'autre part, de détecter les termes pertinents pour un domaine donné. Un terme qui est commun à beaucoup de documents est en effet moins utile pour la recherche qu'un terme qui n'apparaît que dans peu de documents. Ce constat a motivé la combinaison des `TF` et `IDF`. En définitive, le `TF` mesure l'importance d'un terme dans un document tandis que l'`IDF` mesure sa spécificité dans toute la collection de documents c'est-à-dire le corpus.

Analyse distributionnelle d'un corpus

Si l'analyse statistique des *termes/candidats termes* d'un corpus étudie les mots dans leur globalité, l'analyse distributionnelle, pour sa part, propose d'étudier chaque occurrence des *termes/candidats termes* selon leur contexte d'apparition dans le texte (1.9). En effet, comme illustré par la figure 1.10, les *termes/candidats termes* complexes (encore appelés syntagmes) peuvent être décomposés suivant un réseau dit *tête/expansion*. Il s'agit d'une mise en oeuvre du principe de l'analyse distributionnelle "à la Harris" [Harris, 1951].

Parmi les paramètres d'analyse distributionnelle fréquemment utilisés, nous pouvons citer :

1. *la productivité en tête* qui représente le nombre de descendants en tête, c'est-à-dire le nombre de syntagmes dont le *terme/candidat terme* est *tête* ;
2. *la productivité en expansion* qui représente le nombre de descendants en expansion, c'est-à-dire le nombre de syntagmes dont le *terme/candidat terme* est *expansion* ;
3. *le nombre d'ascendants d'un terme/candidat terme* qui représente le nombre de termes/candidats termes qui ont ce *terme/candidat terme* comme descendant (en *tête* ou en *expansion*) ;
4. etc.

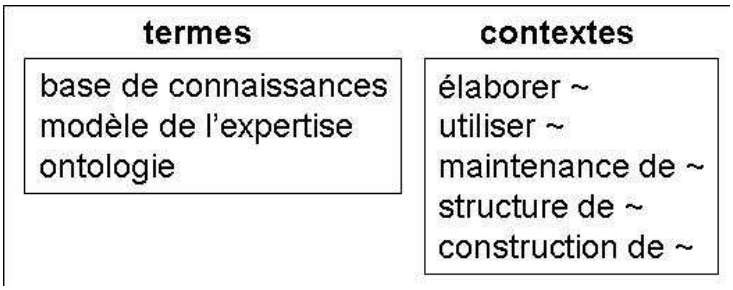

FIGURE 1.9 – Exemple de contexte de termes.

De manière générale, il est important de noter qu'aussi bien pour l'analyse distributionnelle que pour l'analyse statistique, les paramètres étudiés diffèrent selon l'outil de TAL utilisé, certains outils ne se limitant qu'au calcul de quelques paramètres bien précis.

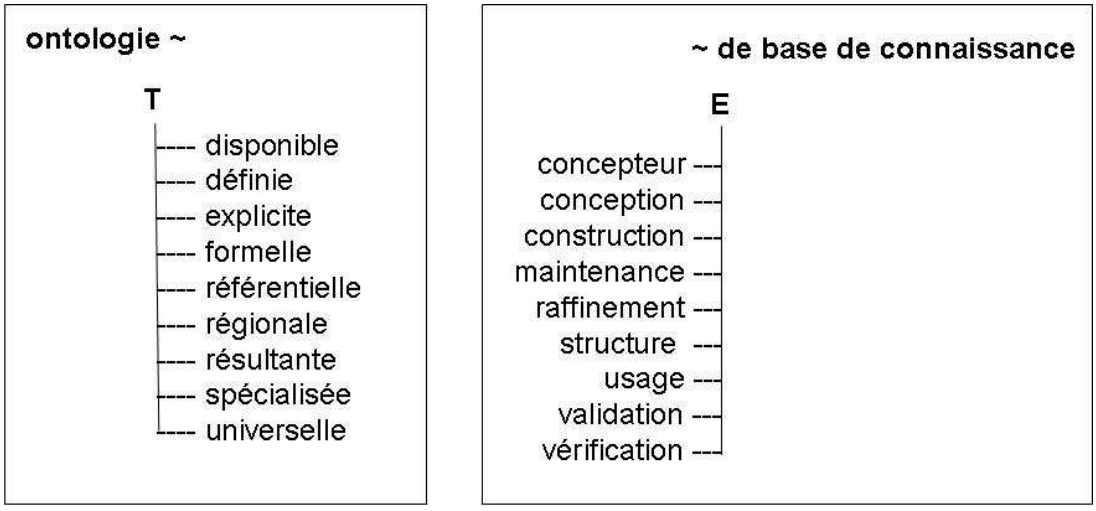

FIGURE 1.10 – Exemple de réseau tête (T) / expansion (E).

1.5.3 Text2Onto

Développé par l'AIFB [14], Text2Onto est un outil de construction d'ontologies qui implémente des algorithmes de fouille de textes sur des corpus textuels pour construire des ontologies de manière semi-automatique [Cimiano and Völker, 2005]. Cet outil inclut plusieurs traitements comme, par exemple, l'extraction des termes en utilisant soit des calculs statistiques, soit des expressions régulières ou encore par identification des relations à l'aide de patrons lexico-syntaxiques ou de calculs de proximités.

1.5.4 Terminae

Développée par le LIPN , TERMINAE [15] est à la fois une méthode et un outil d'aide à l'élaboration de ressources terminologiques et ontologiques à partir de textes [Nathalie et al., 2008]. Cet outil intègre un environnement d'étude terminologique, un environnement d'aide à la conceptualisation et un système de gestion d'ontologies. Tout comme Text2Onto, TERMINAE conserve le lien vers le corpus. Ce lien permet de rendre compte de phénomènes linguistiques tels que la polysémie et la synonymie, et de conserver une trace des choix de l'ontologue pour l'organisation de la hiérarchie de l'ontologie.

1.5.5 OntoCASE

OntoCASE [Blomqvist, 2009] est un outil de conception supervisée d'ontologies impliquant un processus de transformation d'un modèle de connaissances semi-formel en ontologie. OntoCASE s'appuie sur une démarche d'Ingénierie Dirigée par les Modèles. Contrairement à Text2Onto et Terminae où les étapes de construction d'ontologies sont reliées entre elles par des programmes dits *"codés en dur"*, OntoCASE utilise des règles de transformation décrites au travers d'une ontologie dite de transformation écrite en OWL-SWRL [Horrocks et al., 2004].

14. http://www.aifb.uni-karlsruhe.de
15. http://lipn.univ-paris13.fr/terminae/index.php/Main_Page

1.5.6 Construction coopérative d'ontologies

Les outils de cette catégorie prennent acte du fait qu'une ontologie doit faire l'objet d'un consensus et être acceptée par sa communauté d'utilisateurs. Elles adoptent donc une approche collaborative de construction incluant l'intervention de personnes localisées à des endroits différents. Euzenat [Euzenat, 1995, Euzenat, 1996] a identifié un certain nombre de problèmes que pose la construction d'ontologies dans un contexte distribué : la gestion de l'interaction et la communication entre les différentes personnes ; le contrôle de l'accès aux données ; la reconnaissance de l'attribution des droits d'auteurs ; la détection et la correction d'erreurs. En réalité, il ne s'agit pas d'une méthode au sens des deux catégories présentées plus haut, mais plutôt d'un ensemble de pratiques favorisant une construction consensuelle. Comme exemples d'implantation de cette approche, citons celle de [Farquhar et al., 1995] qui a été utilisée pour la construction d'ontologies pour l'intégration de données, celle utilisée à l'INRIA pour la construction de bases de connaissances (CO_4 [Euzenat, 1996]), celle élaborée dans le cadre de l'initiative KA [Benjamins et al., 1999] et utilisée pour la construction d'ontologies pour l'annotation sémantique de documents.

1.6 Conclusion

Dans ce chapitre, nous avons présenté les notions de terminologie, de thésaurus et d'ontologie ; ceci a permis d'appréhender les relations existantes entre ces différentes notions et de voir comment on pourrait transformer une terminologie pour en faire une ontologie. A la base, on dispose d'une liste de termes. Ensuite, on donne un sens précis à ces termes et on obtient une terminologie (ou glossaire). Le thésaurus est obtenu en associant des relations sémantiques et une structuration des termes du glossaire. Enfin, le réseau termino-conceptuel ainsi obtenu peut être structuré et formalisé dans un modèle d'ontologies pour devenir une ontologie formelle. Nous avons également vu, au travers des diverses méthodologies et outils, que la construction d'une ontologie n'est pas une tâche facile. A coté des éditeurs d'ontologies tels que Protégé, PLIBEditor, OntoEdit et DOE qui supposent l'ontologie préalablement conçue sur papier, on trouve des outils d'aide à la construction d'ontologies tels que Terminae et Text2Onto, qui utilisent par exemple des textes pour construire des ontologies. Si ces outils proposent un processus de construction en étapes, le passage d'une étape à l'autre est cependant figé car basé sur une représentation spécifique au sein d'un programme *"codé en dur"*. Certains outils (notamment OntoCASE), par contre, utilisent les transformations de modèles et offrent de ce fait plus de souplesse pour l'amélioration du processus de construction. Dans le chapitre suivant, nous présentons en détail la notion de transformation de modèles ainsi que son utilisation.

Chapitre 2

Transformation de modèles

Sommaire

Résumé. Dans ce chapitre, nous illustrons, d'une part, la problématique que visent à résoudre les transformations de modèles, et, d'autre part, nous discutons des approches de stockage des transformations dans une base de données ainsi que des mécanismes de manipulation et d'exploitation de ces transformations.

2.1 Introduction

La transformation de modèles est un concept central dans les approches de développement piloté par les modèles. Elle fournit un mécanisme pour l'automatisation de la manipulation de modèles. Dans ces approches de développement, les modèles sont créés, améliorés et maintenus. Dans le cas de la conception logicielle, par exemple, les modèles sont des artefacts primaires du développement et contiennent des informations à différentes étapes du cycle de vie du logiciel. Parmi ces informations on peut citer des exigences telles que l'analyse, la conception, l'implémentation, le test, la qualité, la simulation, la vérification, etc.. Les transformations de modèles peuvent également être utilisées pour fournir un mécanisme permettant de créer et de mettre à jour automatiquement des données d'un ou plusieurs modèles sur la base des données contenues dans d'autres modèles. Dans le cas de l'échange de données, par exemple, une transformation de modèles bien définie peut être utilisée pour garantir la cohérence des données d'un modèle par rapport aux modifications des données d'un autre modèle.

Dans ce chapitre, nous fixons, dans la section 2.2, le vocabulaire lié aux transformations de modèles telles qu'elles sont utilisées dans la suite de ce document. Ensuite, nous décrivons, dans la section 2.3, quelques scénarios usuels pour lesquels la transformation des modèles est utilisée. Dans la section 2.4, nous présentons les différents paradigmes ou approches de transformation de modèles. Dans les sections 2.7 et 2.8 , nous présentons respectivement les problématiques liées à la persistance et à l'exploitation des transformations. Dans la section 2.9, nous présentons l'implémentation des transformations de modèles dans l'outil OntoCASE.

2.2 Terminologie

Dans cette section, nous précisons, pour le domaine de la transformation des modèles, le vocabulaire utilisé dans cette thèse.

1. *Modèle.* Un modèle est une représentation d'un système permettant une meilleure compréhension de ce système [Bézivin and Gerbé, 2001, Seidewitz, 2003]. Le modèle d'un domaine peut être décrit soit en utilisant un langage spécifique au domaine, soit en utilisant un langage de modélisation indépendant du domaine (UML par exemple).

2. *Transformation de modèles*. Plusieurs définitions ont été proposées pour la notion de transformation de modèles. Selon Tratt [Tratt, 2005], une transformation de modèles est "*un programme qui convertit un modèle en un autre modèle*". Pour L'OMG [16] et le monde de l'ingénierie du logiciel qui s'intéressent au contexte du Model-Driven Architecture, *"la transformation de modèles est un processus de conversion d'un modèle en un autre modèle du même système"* [Miller and Mukerji, 2003]. Pour Kleppe [Kleppe et al., 2003], *"la transformation de modèle consiste en la génération automatique d'un modèle cible à partir d'un modèle source conformément à la description de la transformation"*. Mens [Mens and Vangorp, 2006] étend la définition proposée par Kleppe en offrant la possibilité d'avoir plusieurs modèles en entrée comme en sortie d'une transformation.

16. OMG : Object Management Group. http://www.omg.org/

Pour les besoins du projet DaFOE4App, la définition que nous retenons est proche de celle proposée par Kleppe [Kleppe et al., 2003]. Ainsi, dans ce mémoire, la transformation de modèle consiste, en la description des similarités conceptuelles entre deux modèles dans le but de pouvoir, à partir d'une population d'instances du modèle source, créer automatiquement une population d'instances du modèle cible. De plus, une transformation entre un modèle source et un modèle cible doit permettre, à partir du modèle cible, d'interroger le modèle source.

La figure 2.1 illustre les principales composantes d'un processus de transformation.

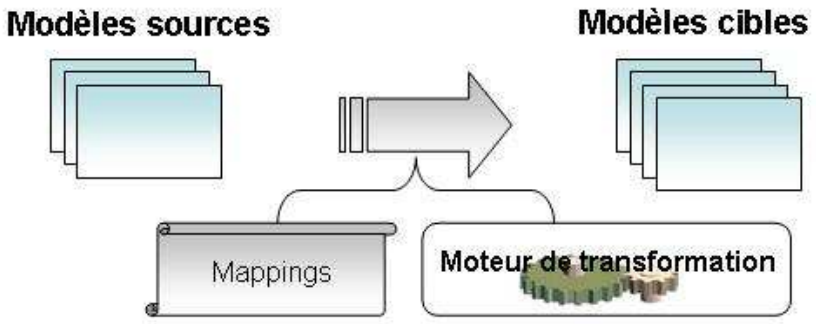

FIGURE 2.1 – Principales composantes d'un processus de transformation.

3. *Méta-modèle.* Un méta-modèle d'un modèle `m` décrit la structure que doit respecter le modèle `m` afin d'être valide. Un méta-modèle est comparable à la grammaire d'un langage.

4. *Méta-méta-modèle*. Un méta-méta-modèle d'un modèle représente le méta-modèle permettant de décrire le méta-modèle du modèle `m`. Dans le cas des langages, un méta-méta-modèle est comparable à la grammaire (EBNF par exemple) utilisée pour construire la grammaire du langage. Parmi les méta-méta-modèles standards, nous pouvons citer le méta-méta-modèle MOF[17] proposé par l'OMG et utilisé pour la modélisation objet ; le modèle Entité-Association [Chen, 1976] utilisé dans le domaine des bases de données relationnelles, etc..

5. *Langage de transformation de modèles*. Un langage de transformation de modèles est un vocabulaire doté d'une sémantique pour la transformation de modèles. Une transformation de modèle peut également être matérialisée par des modèles. On parle à ce moment de modèle de transformation.

6. *Mapping*. Encore appelée *description de la transformation* ou tout simplement *transformation*, un mapping exprime un appariement entre les éléments d'un modèle source et ceux d'un modèle cible. Ce mapping est écrit soit au moyen d'un langage de transformation, soit en utilisant un modèle de transformation ou modèle de mapping. Dans le cadre de nos travaux, un mapping d'un modèle source vers une modèle cible est utilisé, par exemple, pour extraire à la volée les instances du modèle source respectant des contraintes spécifiées sur le modèle cible. De plus, compte tenu

17. Meta Object Facilty: http://www.omg.org/mof/

de la structure des modèles, les mappings peuvent être classifiés selon une granularité plus fine d'éléments à apparier. On parle dans ce cas de *constructeur de mappings* pour désigner l'unité atomique d'un mapping. En d'autres termes, un constructeur de mappings permet d'apparier les éléments des modèles.

7. *Moteur de transformation*. Un moteur de transformation est un outil chargé d'interpréter et/ou d'exécuter des mappings.

La figure 2.2 résume les différents niveaux d'abstraction d'un système à base de mappings.

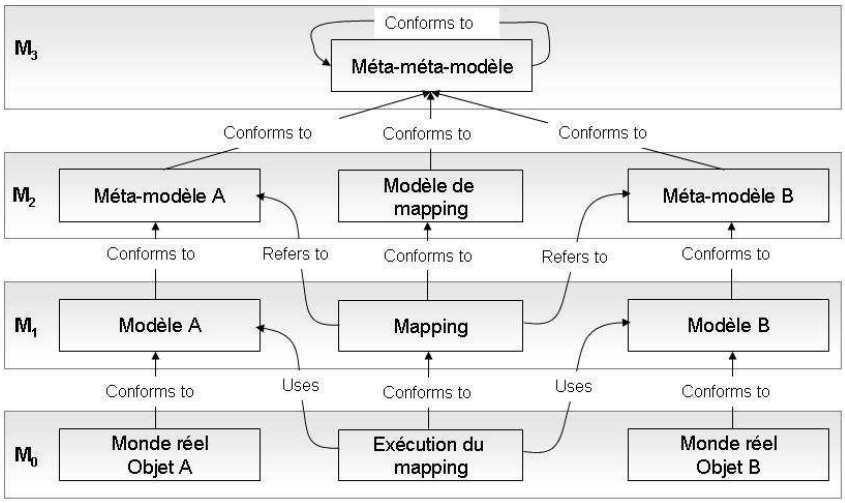

FIGURE 2.2 – Niveaux d'abstraction d'un processus de transformation de modèles.

2.3 Intérêt de la transformation de modèles

La transformation de modèles est applicable de plusieurs manières dans un contexte d'Ingénierie Dirigée par les Modèles. Elle s'utilise dans des scénarios tels que la création de modèles, la modification de modèles, l'adaptation de modèles, la fusion de modèles, le partage de données, l'intégration de données, etc.. Au cœur de tous ces scénarios se trouve le besoin de réutiliser soit les modèles, soit les instances des modèles. Dans cette thèse, nous nous intéressons uniquement à la transformation de modèles comme solution au problème de médiation d'instances **??**. Par médiation d'instances, nous entendons le calcul, à la volée, d'une instance cible (conforme à un modèle cible) à partir d'une instance source (conforme à un modèle source) en utilisant le mapping existant entre les modèles source et cible. De manière générale, et comme l'illustre la figure 2.3, la médiation d'instances s'effectue par un moteur de transformation qui, à partir d'un mapping entre un modèle source et un modèle cible, construit une instance cible à partir d'une instance source.

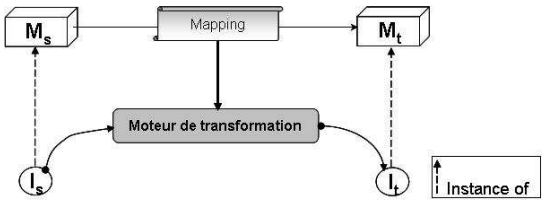

FIGURE 2.3 – Médiation d'instances.

2.4 Approches de transformation de modèles

Un paradigme ou approche de transformation de modèles est le principe de conception sur lequel s'appuie la représentation des mappings. Plusieurs outils proposent un environnement pour la transformation de modèles. En d'autres termes, un paradigme de transformation de modèles décrit la façon de coder les mappings. Dans cette section, nous les regroupons selon qu'ils favorisent ou non l'évolution des modèles participants aux mappings.

2.4.1 Approche Impérative

Dans cette approche, les mappings sont *"codés en dur"* dans un programme. Le mapping est un programme écrit à l'aide des langages de programmation tels que C/C++, Java, SQL, etc. Le mapping est donc représenté par une suite d'actions à exécuter. Si cette approche présente l'avantage d'être facile à implémenter, elle présente cependant l'inconvénient d'être figée du point de vue des modèles impliqués dans le mapping. En effet, toute modification faite sur l'un des modèles impliqués dans le mapping nécessite, d'une part, la réécriture du mapping, c'est-à-dire le programme, et, d'autre part, la recompilation complète de ce programme.

2.4.2 Approche Déclarative

A partir des années 2000, Berstein [Bernstein, 2003], au travers de la notion de *"Model Management"*, a proposé des opérateurs pour la manipulation des modèles. Ces opérateurs de manipulation des modèles ont été précurseurs de l'idée de modéliser les mappings : on parle alors d'approche déclarative ou par méta-modélisation. Les mappings sont alors considérés comme des instances d'un modèle (appelé modèle de mapping) et peuvent de ce fait être manipulés comme des objets. En interrogeant le modèle de mapping, on obtient par exemple, la description des mappings. Enrichir la transformation entre deux modèles se résume donc à créer de nouvelles instances du modèle de mapping. L'avantage ici est de pouvoir le faire dynamiquement, c'est-à-dire sans recompilation des programmes utilisant les mappings. On peut donc dire que, contrairement à l'approche impérative qui décrit la manière d'exécuter les mappings (le mapping étant un programme), l'approche par méta-modélisation s'intéresse davantage à décrire des appariements établissant une équivalence conceptuelle entre les modèles. De plus, dans cette approche, les mappings peuvent être bidirectionnels, permettant ainsi de transformer les instances d'un modèle A vers un modèle B et inversement.

Remarque. Certains environnements de transformation de modèles implémentent à la fois l'approche

impérative et déclarative. On parle dans ce cas d'approche hybride car elle offre à l'utilisateur la possibilité d'utiliser l'approche de son choix ou une combinaison des deux approches.

2.5 Traçabilité des transformations

Dans le processus de transformation, la traçabilité peut être produite comme un effet secondaire à l'exécution des mappings. La traçabilité présente, d'une part, une vue d'ensemble sur les mappings, et, d'autre part, fournit un historique de l'exécution des mappings. La traçabilité renseigne sur les éléments du modèle source qui ont été appariés à des éléments du modèle cible. La fonctionnalité de traçage peut être intégrée dans l'outil ou mise en œuvre dans le cadre de la description du mapping. Les traces peuvent être stockées dans le modèle source, dans le modèle cible ou dans un endroit séparé.

Pour garantir la traçabilité des mappings, les langages de transformations offrent la possibilité de définir un sens d'exécution de la transformation. De ce fait, on distingue :

- les langages *unidirectionnels* qui interprètent le mapping dans un seul sens (du modèle source vers le modèle cible) ;
- les langages *bidirectionnels* pour lesquels les notions de modèles source et cible sont relatives au sens d'interprétation du mapping. En effet, un même mapping étant utilisé dans les deux cas d'interprétation, un modèle sera source s'il est en entrée dans l'interprétation du mapping. Dans le cas contraire, il est dit cible.

Certains mappings sont dits approximatifs, c'est-à-dire qu'ils traduisent une similarité *subjective* d'une interprétation à l'autre. La logique floue est en général utilisée dans cette situation.

2.6 Logique floue dans les transformations

Dans la représentation des mappings, on peut vouloir, exprimer la pertinence d'un mapping par rapport aux autres mappings. L'approche couramment utilisée dans la littérature consiste à affecter, à chaque mapping, une valeur appelée *degré de confiance* et comprise entre `0` et `1`. Un mapping est d'autant plus pertinent que son degré de confiance est voisin de `1`. Dans la pratique, la logique floue est en général implémentée suivant une approche probabiliste ou à l'aide des ensembles flous.

2.6.1 Approche possibiliste

Dans travaux portant sur les bases de données dites probabilistes [Fuhr and Rölleke, 1997], à chaque tuple est associé une probabilité d'appartenir à une table. La figure 2.4 (a) présente une base de données probabiliste $\mathcal{D}^p$ constituée de deux tables $\mathcal{S}^p$ et $\mathcal{T}^p$ [18]. Les deux tuples de $\mathcal{S}^p$ possèdent les probabilités respectives `0.8` et `0.5` tandis que le tuple de $\mathcal{T}^p$ possède la probabilité `0.6`. Pour simplifier, on suppose que les tuples sont indépendants (au sens des probabilités). La notion de base de données probabiliste traduit l'existence d'une distribution de probabilité sur les instances de la base de données regroupés

18. l'exposant "p" signifie probabiliste et est utilisé en guise de notation

dans ce que Dalvi et Suciu [Dalvi and Suciu, 2007] appellent ensemble de tuples possibles (ou *possible worlds*) et notent $pwd(\mathcal{S}^p)$. La figure 2.4 (b) présente les huit tuples qu'il est possible de construire à partir de $\mathcal{S}^p$ et $\mathcal{T}^p$. La probabilité du tuple résultant est calculée en multipliant les probabilités des tuples car nous avons supposé que les probabilités des tuples sont indépendantes. Par exemple, la probabilité du tuple D_2 vaut $0.8 \times (1 - 0.5) \times 0.6 = 0.24$ car ce tuple contient les tuples s_1 et t_1 et ne contient pas le tuple s_2. De manière générale, soit T, un tuple de $\mathcal{D}^p$, la probabilité de ce tuple, notée p(T) est définie par :

$$p(T) = \prod_{t \in (tuple(\mathcal{S}^p) \cup tuple(\mathcal{T}^p))} \mu(t, T) \text{ avec :}$$

– $\mu(t, T) = \begin{cases} \bullet\ p(t) \textit{ si } T \textit{ contient le tuple } t\ ; \\ \bullet\ 1 - p(t) \textit{ sinon.} \end{cases}$

– *tuple* est une fonction qui retourne les tuples d'une table donnée.

S^p

oid	A	B	Probabilité
s_1	'm'	1	0.8
s_2	'n'	1	0.5

D^p

oid	C	D	Probabilité
t_1	1	'p'	0.6

(a)

$Pwd(D^p)$

Instances	Probabilités
$D_1 = \{s_1, s_2, t_1\}$	0.24
$D_2 = \{s_1, t_1\}$	0.24
$D_3 = \{s_2, t_1\}$	0.06
$D_4 = \{t_1\}$	0.06
$D_5 = \{s_1, s_2\}$	0.16
$D_6 = \{s_1\}$	0.16
$D_7 = \{s_2\}$	0.04
$D_8 = \{\}$	0.04

(b)

FIGURE 2.4 – (a) Une base de données probabiliste $\mathcal{D}^p$, (b) ensemble des tuples possibles pour $\mathcal{D}^p$.

En utilisant l'approche probabiliste, des probabilités sont donc affectées aux mappings. La base de données probabiliste devient dans ce cas une base de données dont les instances sont les mappings et le calcul de probabilité d'un tuple correspond au calcul du degré de confiance résultant de la composition des mappings.

2.6.2 Approche utilisant les ensembles flous

Dans cette approche, le degré de confiance ne représente pas une probabilité mais un poids, en général compris entre 0 et 1. Un degré de confiance peut aussi correspondre à un ensemble ordonné et discret de

valeurs. En effet, selon la théorie des ensembles flous [Zadeh, 1965], une partie floue A d'un ensemble E est caractérisée par une application de E dans [0, 1]. Cette application, appelée *fonction d'appartenance* et notée μ_A représente le degré de validité de la proposition *"x appartient à A"* pour chacun des éléments x de E. Si $\mu_A(x) = 1$, l'objet x appartient totalement à A, et si $\mu_A(x) = 0$, il ne lui appartient pas du tout. Pour un élément x donné, la valeur de la fonction d'appartenance $\mu_A(x)$ est appelée *degré d'appartenance* de l'élément x au sous-ensemble A.

Une partie floue A d'un ensemble E peut aussi être caractérisée par l'ensemble de ses α-coupes. Une α-coupe d'une partie floue A est le sous-ensemble des éléments ayant un degré d'appartenance supérieur ou égal à α i.e $\alpha - \text{coupe}(A) = \{x \in E \mid \mu_A(x) \geq \alpha\}$.

2.7 Persistance des transformations

Dans les sections précédentes, nous avons présenté de façon générale le problème de la transformation de modèles. Cette présentation s'intéressait à la description des concepts portant sur les mappings ainsi qu'à l'illustration de la problématique liée au mapping. Nous n'avons pas évoqué les aspects liés au besoin de persistance des mappings. En effet, les mappings gérés en mémoire centrale peuvent poser un problème de performance et de traçabilité. Par traçabilité, nous entendons, d'une part, la possibilité de connaître à tout instant, les mappings existants entre des modèles, et, d'autre part, nous entendons la capacité à interroger ces mappings, les modifier, etc.. De ce fait les mappings gérés en mémoire centrale sont à la fois difficilement réutilisables et dynamiquement modifiables. Dans la pratique, la persistance des mappings dépend aussi de la nature des modèles (schémas XML, schéma de base de données, etc.) à mapper. Dans cette section, nous nous intéressons au cas particulier des schémas de base de données, et, nous discutons des capacités des architectures de base de données de type méta-modélisation (base de données à base ontologique) à gérer efficacement la persistance des mappings.

2.7.1 Base de Données à Base Ontologique (BDBO)

L'objectif des BDBO est de disposer, au sein d'une base de données, d'une sémantique décrivant les données contenues dans la base de données. Cette sémantique, portée par des ontologies, est ensuite utilisée dans divers domaine d'application tels que l'intégration de données, le partage de données, etc..

Au cours des dernières années, plusieurs architectures de BDBO ont été proposées. Nous présentons tout d'abord une taxinomie de ces architectures. Puis, nous discutons de leurs capacités à résoudre les problèmes de passage à l'échelle et de gestion des mappings qui se posent dans le contexte du web sémantique en général et dans le cas de la plate-forme DaFOE en particulier.

2.7.1.1 Caractéristiques

Une base de données à base ontologique est une base de données qui possède deux caractéristiques principales :

1. les ontologies et leurs instances sont stockées dans la base de données et peuvent faire l'objet des mêmes traitements (insertion, mise à jour, suppression, etc.). Par la suite, nous utilisons le terme

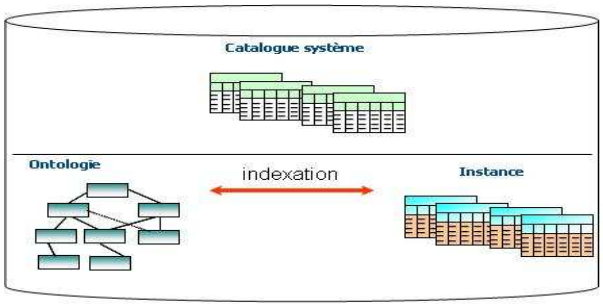

FIGURE 2.5 – Architecture Générale d'une BDBO.

données de niveau modèle pour désigner les ontologies (instance d'un méta-modèle) et le terme *données de niveau instance* pour désigner les instances des ontologies ;

2. toute *donnée de niveau instance* est associée à une *donnée de niveau modèle* qui en définit le sens.

2.7.1.2 Classification

D'après la classification proposée par [Fankam, 2009], on distingue trois types de BDBO. Ces types de BDBO se différencient essentiellement par le nombre de schéma utilisés pour représenter les ontologies et leurs instances.

BDBO de type 1

Dans les BDBO de type 1, les données sont représentées en utilisant un seul schéma composé d'une unique table de triplets `(subject, predicate, object)` [Harris and Gibbins, 2003, Chong et al., 2005, Petrini and Risch, 2007, Wilkinson et al., 2003]. Cette table, qualifiée de table verticale [Agrawal et al., 2001], peut être utilisée à la fois pour les données de niveau modèle et de niveau instance. Pour les données de niveau modèle, les trois colonnes de cette table représentent respectivement l'identifiant d'un élément de l'ontologie, un prédicat et la valeur du prédicat qui peut être soit un identifiant d'un élément de l'ontologie, soit une valeur littérale. Par exemple, le triplet [19] `(Student, subClassOf, Person)` représente une relation de subsomption entre les classes `Student` et `Person`. Pour les données de niveau instance, les trois colonnes de cette table représentent respectivement l'identifiant d'une instance, une caractéristique d'une instance (c'est-à-dire, une propriété ou l'appartenance à une classe) et la valeur de cette caractéristique. Par exemple, le triplet `(Peter, grade, PhD)` représente le fait que Peter a le grade de Docteur. Pour illustrer les différentes approches de stockage proposées par les BDBO, nous utilisons un sous-ensemble d'une ontologie du domaine universitaire (Cf. figure 2.6). Un extrait de la table verticale correspondante est illustré par la figure 2.7.

A cause de l'utilisation d'une table unique, les BDBO de type 1 posent des problèmes de performance lorsque des requêtes requièrent de nombreuses auto-jointures sur cette table [Alexaki et al., 2001]. Pour améliorer les performances, chaque colonne de la table verticale doit être indexée [Ma et al., 2004]. De plus, la colonne qui correspond au prédicat doit être clusterisée [Agrawal et al., 2001] ou des vues

19. RDF utilise des URI comme identifiants. Pour des raisons de lisibilité nous utilisons des noms.

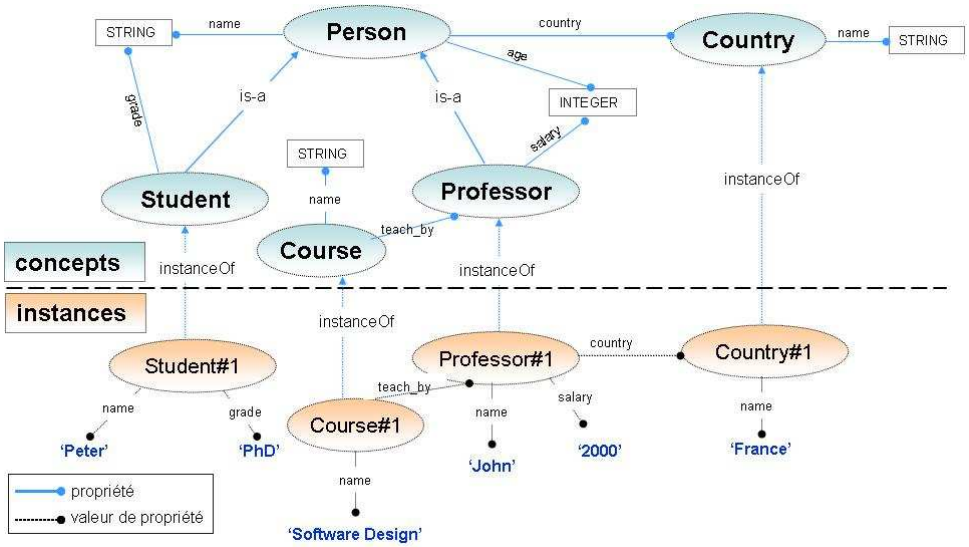

FIGURE 2.6 – Ontologie du domaine universitaire.

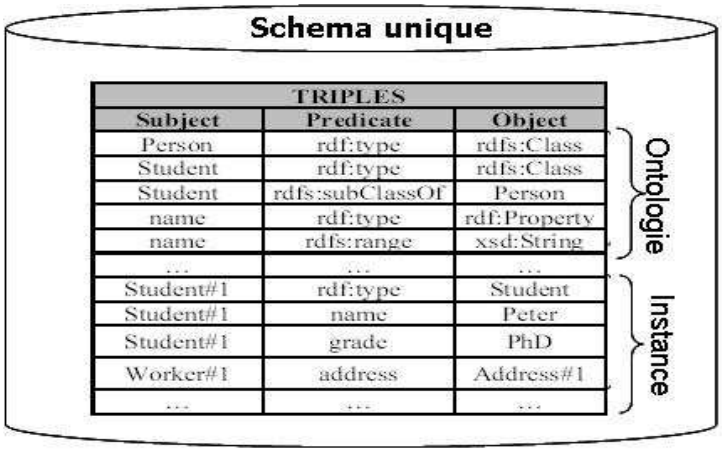

FIGURE 2.7 – BDBO de type 1.

matérialisées doivent être créées [Pan and Heflin, 2003]. Dans les deux cas, cette approche entraîne un coût de stockage supplémentaire et un surcoût de traitement des opérations de mises à jour. [Alexaki et al., 2001, Ma et al., 2004, Theoharis et al., 2005] ont montré que dans de multiple cas, les BDBO de type 2 surpassent les BDBO de type 1 malgré l'ajout des techniques d'optimisation faites sur ces dernières. L'infrastructure proposée par Oracle [Chong et al., 2005] pour gérer simultanément les tables de données usuelles et une table de triplets RDF/RDFS suit l'approche de type 1. Cependant, elle offre tous les mécanismes nécessaires pour matérialiser des vues pouvant correspondre, en particulier, à une structure de type 2 ; ce qui lui permet, sous réserve d'optimisation utilisant des vues et des index par l'administrateur de la base de données, d'avoir l'efficacité des BDBO de type 2.

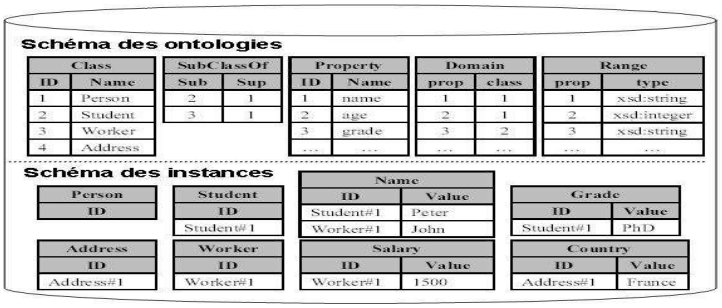

FIGURE 2.8 – BDBO de type 2.

BDBO de type 2

Les BDBO de type 2 stockent séparément les données de niveau modèle (les ontologies) et les données de niveau instance dans deux schémas distincts [Alexaki et al., 2001, Broekstra et al., 2002, Ma et al., 2004]. Le schéma dédié au stockage les données de niveau modèle dépend du méta-modèle (modèle d'ontologies) utilisé pour représenter les ontologies. Ce schéma est composé de tables utilisées pour stocker chaque concept du modèle (classes, propriétés, relations de subsomption, etc.). Pour les données de niveau instance, plusieurs schémas ont été proposés. Une table verticale peut être utilisée pour stocker les données de niveau instance sous forme de triplets [Broekstra et al., 2002, Ma et al., 2004]. Une alternative consiste à utiliser une représentation binaire dans laquelle chaque classe est représentée par une table unaire et chaque propriété par une table binaire [Abadi et al., 2007, Alexaki et al., 2001, Broekstra et al., 2002, Pan and Heflin, 2003]. Une autre approche, appelée table par classe, consiste à associer, à chaque classe, une table d'instances dont chacune des colonnes correspond à une propriété pour laquelle au moins une instance de la classe doit posséder une valeur [Park et al., 2007, Dehainsala et al., 2007a]. Ces trois principales approches ont également des variantes (voir [Theoharis et al., 2005] pour plus de détails). La figure 2.8 présente un exemple de BDBO de type 2 stockant les données de l'exemple précédent (Cf. figure 2.6). Dans cet exemple, les données de niveau modèle sont stockées en utilisant un schéma pour des ontologies RDFS et les données de niveau instance sont représentées par une représentation binaire.

Les performances des BDBO de type 2 dépendent de la représentation utilisée pour le stockage *données de niveau instance*. Les évaluations de performance conduites dans [Agrawal et al., 2001, Theoharis et al., 2005, Pan and Heflin, 2003, Abadi et al., 2007] ont montré que l'approche utilisant une table verticale pour les instances souffre des mêmes faiblesses que celles évoquées précédemment pour les BDBO de type 1. Ainsi, la représentation binaire a été considérée pendant longtemps comme la meilleure représentation pour les instances. Cependant, les résultats expérimentaux obtenus sur la représentation table par classe ont remis en cause cette idée [Park et al., 2007, Dehainsala et al., 2007a]. Pour les requêtes où les classes à interroger sont spécifiées, la représentation table par classe surpasse l'approche binaire par un ratio souvent supérieur à 10, en particulier lorsque les instances sont associées à plusieurs propriétés [Dehainsala et al., 2007a]. De plus, les insertions et mises à jour sont plus rapides. Le seul cas où la représentation binaire est meilleure que la représentation table par classe correspond aux requêtes où la classe à interroger n'est pas spécifiée et qui ne concernent qu'un nombre faible de propriétés.

BDBO de type 3

Les BDBO de type 3 [Pierra et al., 2005, Dehainsala et al., 2007a] proposent l'ajout d'un schéma supplémentaire aux BDBO de type 2. Ce schéma, appelé méta-schéma (ou méta-méta-modèle), stocke le méta-modèle (modèle d'ontologies) dans un méta-méta-modèle réflexif. Pour le méta-modèle, le méta-schéma joue le même rôle que celui joué par la méta-base (ou catalogue système) dans les bases de données classiques. En effet, le méta-schéma permet :

- un accès générique aux ontologies ;
- l'évolution du modèle d'ontologies utilisé ;
- le stockage de plusieurs modèles d'ontologies.

Notons que l'architecture proposée par les BDBO de type 3 est semblable aux différentes couches d'abstraction de l'architecture du MOF dans laquelle la partie méta-schéma correspond au niveau $\mathtt{M}_3$, la partie schéma correspond au niveau $\mathtt{M}_2$ et la partie instance représente les niveaux $\mathtt{M}_1$ et $\mathtt{M}_0$.

En guise d'illustration de ce type de BDBO, nous présentons, dans la section suivante, la BDBO OntoDB sur laquelle s'appuie notre travail.

2.7.2 La BDBO OntoDB

OntoDB [Dehainsala et al., 2007a] est une BDBO de type 3 qui permet de représenter les données et les ontologies dans des schémas séparés tout en proposant une structure permettant d'accéder, de stocker et de faire évoluer le modèle d'ontologie utilisé. La représentation de cette structure, nommée méta-schéma, permet d'adapter OntoDB aux évolutions du modèle d'ontologies. La figure 2.9 présente les quatre parties qui composent la BDBO OntoDB :

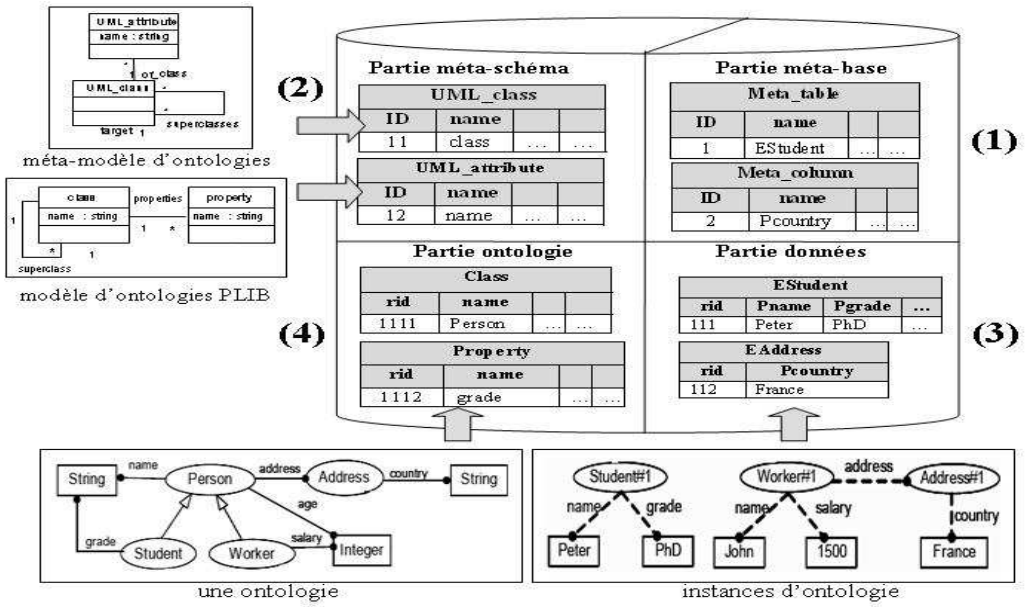

FIGURE 2.9 – Architecture d'OntoDB.

- **la partie *méta-base* (1).** La *métabase*, souvent appelée *system catalog*, est une partie traditionnelle des bases de données. Elle est constituée de l'ensemble des tables systèmes. Ces tables sont celles dont le SGBD se sert pour gérer et assurer le fonctionnement de l'ensemble des données contenues dans la base de données. Dans OntoDB, elles contiennent, en particulier, la description de toutes les tables et colonnes définies dans les trois autres parties de cette architecture ;

- **la partie *données* (3).** Elle représente les objets du domaine, décrits en termes d'une classe d'appartenance et d'un ensemble de valeurs de propriétés applicables à cette classe. Ces objets sont représentés en associant une table à chaque classe (représentation horizontale) ;

- **la partie *ontologie* (4).** Elle contient les ontologies définissant la sémantique des différents domaines couverts par la base de données. Ce prototype supporte le modèle d'ontologies PLIB ;

- **la partie *méta-schéma* (2).** Elle représente, au sein d'un modèle réflexif, à la fois le modèle d'ontologies utilisé et le *méta-schéma* lui-même. Le *méta-schéma* représente, pour la partie *ontologie*, ce qu'est la *méta-base* pour la partie données.

La BDBO OntoDB est également équipée d'un langage de requête nommé OntoQL [Jean et al., 2005, Jean et al., 2006, Jean et al., 2007] permettant de manipuler séparément les parties méta-schéma, ontologie et instance mais aussi de manipuler, conjointement au sein d'une seule requête, les parties ontologie et instance. Par convention OntoQL utilise le préfixe `"#"` pour différencier, à l'intérieur d'une requête les données de niveau modèle et celles de niveau instance.

Exemple. **Requête OntoQL**

Retourner les classes contenues dans la BDBO avec leurs propriétés applicables et les identifiants de leurs instances.

```
SELECT c.#name, p.#name, i.oid
FROM #Class AS c, UNNEST(c.#properties) AS p, c AS i
```

Dans cette requête, l'itérateur `c` parcourt les classes. Un itérateur dynamique `i` est introduit sur les instances de ces classes par la syntaxe `c AS i`. La valeur de la propriété `oid` est retournée pour chacune de ces instances.

Notons cependant que l'ajout du méta-schéma, dans les BDBO de type 3, n'améliore pas les performances des requêtes mais offre, comme nous l'avons précisé précédemment, de nouvelles fonctionnalités. Dans le cadre de notre thèse, nous montrons que la disponibilité de ce méta-schéma dans la base de données facilite, en particulier, l'extension du méta-modèle permettant d'implémenter l'approche de création dynamique des constructeurs de mappings que nous proposons.

Remarque. Dans la suite de ce document, les modèles que nous manipulons ne respectent pas forcément les caractéristiques des ontologies c'est-à-dire que ces modèles ne sont pas forcément consensuels et partageables. Pour cette raison, nous parlons tout simplement de modèle. Ainsi, la notion de base de données à base de modèles (BDBM) sera préférée à celle de base de données à base ontologique. En d'autres termes une BDBM est équivalente à une BDBO à la différence que, la *partie ontologie*

de la BDBO, appelée *partie modèle ou schéma* dans la BDBM, stocke non plus des ontologies en particulier, mais des modèles en général ; une ontologie est en réalité un cas particulier de modèle.

2.8 Exploitation des mappings

La littérature propose plusieurs langages pour la définition, la manipulation et l'interrogation des modèles et des mappings. Dans cette section, nous les regroupons en trois principales catégories dont nous présentons les fonctionnalités et discutons de la nécessité d'un langage complémentaire pour la manipulation des mappings.

2.8.1 Langages de transformation

Certains langages de programmation possèdent une formulation déclarative permettant de décrire des mappings simples tandis que d'autres langages proposent des constructeurs de mappings impératifs permettant d'écrire des mappings plus complexes. Parmi ces langages, on retrouve des standards tels que les langages QVT (Query/View/Transformation) [Kurtev, 2008] et ATL (ATLAS Transformation Language) [Jouault et al., 2008] qui proposent à la fois des constructeurs impératifs et des constructeurs déclaratifs. Le langage Kermeta [Moha et al., 2010], pour sa part, propose un environnement impératif de définition des mappings permettant de spécifier et valider des contraintes, des comportements, etc..

Cette liste non exhaustive de langages de transformation de modèles peut être complétée par des langages tels que ETL (Epsilon Transformation Language) [Kolovos et al., 2008], TCS (Textual Concrete Syntax) [Jouault et al., 2006], Tefkat [Steel and Lawley, 2004], MOLA [Kalnina et al., 2010], etc.

Toutefois, dans cette catégorie de langages, les modèles et méta-modèles doivent être explicitement chargés en mémoire centrale. Par conséquent, il est difficile d'exécuter les mappings de manière incrémentale. De plus, cette catégorie de langages ne fournit pas de sous-ensemble fonctionnel permettant d'interroger et donc de manipuler les modèles et leurs instances en exploitant les mappings.

2.8.2 Langages pour les systèmes à base méta-modélisation

Les systèmes dits à base de méta-modélisation permettent de manipuler les méta-données représentées sous forme de modèles ou de méta-modèles. Pour faciliter la gestion de tels systèmes, des langages de requêtes sont nécessaires. L'approche classique consiste à définir un langage de requêtes basé sur une architecture de méta-modélisation conforme au MOF et offrant la possibilité de manipuler à la fois les données et les méta-données. Parmi les langages conçus pour les systèmes à base de méta-modélisation, nous pouvons citer les langages tels que SchemaSQL [Lakshmanan et al., 2001, Lakshmanan et al., 1999], MSQL [Litwin et al., 1989, Grant et al., 1993], SQL/M [Kelley et al., 1995], mSQL [Petrov and Nemes, 2008] adaptés aux systèmes multibases ou bases de données fédérées. D'autres langages, issues du domaine de la modélisation à base d'ontologies, permettent de manipuler à la fois le modèle d'ontologies, les ontologies et leurs instances. Parmi cette catégorie de langages, nous pouvons citer le langage OntoQL [Jean et al., 2005, Jean et al., 2006, Jean et al., 2007] qui exploite l'architecture de méta-modélisation de la BDBM OntoDB [Dehainsala et al., 2007b], le langage SparQL [Makris et al., 2010]

dédié aux bases de données de type RDF [Manola and Miller, 2004], le langage RQL [Karvounarakis et al., 2002, Karvounarakis et al., 2004], etc..

En général, l'objectif commun à cette catégorie de langage n'est pas la gestion des mappings car très souvent, les mappings n'y sont ni explicitement représentés ni exploités. Cette catégorie de langage facilite plutôt la manipulation de l'architecture de méta-modélisation.

2.8.3 Langages orientés mapping

Les langages orientés mapping sont des langages qui intègrent la notion de mapping et offrent la possibilité d'exploiter les mappings lors de l'interrogation des données. Parmi ces langages, nous pouvons citer le langage SparQL [Makris et al., 2010]. En effet, dans la modélisation d'ontologies avec le langage OWL, les ontologies peuvent être appariées à l'aide des constructeurs de classes d'équivalence par exemple. SparQL permet ensuite à un utilisateur d'interroger le graphe d'ontologies résultant en raisonnant sur les classes appariées.

Il existe également des plate-formes de gestion de modèles tels que Rondo [Melnik et al., 2003a, Melnik et al., 2003b] qui fournissent des opérateurs de haut niveau pour la manipulation des modèles (*diff, compose, merge, etc.*) mais pour lesquelles il manque un langage de requête permettant d'interroger des modèles en raisonnant sur les mappings.

Comme limitation commune à ces langages ou frameworks, nous pouvons dire qu'ils ne permettent pas de paramétrer le processus d'exploitation des mappings. En général, le processus d'exploitation des mappings est implicite et masqué à l'utilisateur. De ce fait, la totalité du graphe de mappings est exploitée même si l'utilisateur ne souhaite utiliser qu'un sous-ensemble bien précis de ce graphe. De plus, dans ces langages ou frameworks, la modélisation des mappings est statique et il n'est pas possible de l'étendre dynamiquement. En effet, du fait que les bases de données sous-jacentes à ces langages ou frameworks ne sont pas des BDBM de type 3, la modélisation des mappings reste figée.

2.9 Transformation de modèles dans OntoCASE

Comme nous l'avons évoqué dans le chapitre précédent (Cf. section 1.5.5), OntoCASE est un outil de construction d'ontologies utilisant les transformations de modèles. Dans cette section, nous présentons brièvement le processus de construction proposé par OntoCASE et décrivons l'utilisation des transformations de modèles dans cet outil.

2.9.1 Méthodologie de construction

OntoCASE permet de construire une ontologie à partir d'un modèle semi-formel. La figure 2.10 qui résume les étapes d'une démarche de construction instrumentée par un assistant informatique, permet de positionner OntoCASE par rapport au processus de construction d'ontologies selon le degré de formalisation des connaissances. On remarque, sur cette figure, qu'OntoCASE démarre la construction à partir de connaissances semi-formelles exprimées dans le langage MOT [20] [Paquette, 2005] pour ensuite

20. Modélisation par Objets Typés

formaliser ces connaissances en une ontologie conforme au langage OWL.

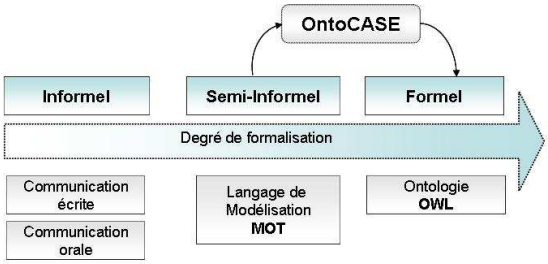

FIGURE 2.10 – Méthodologie OntoCASE.

2.9.2 Transformation de modèles

OntoCASE utilise des ontologies pour représenter les étapes de la construction d'ontologies. De ce fait, les mappings, tels que nous les avons présenté dans ce chapitre, s'appliquent aux ontologies. Cette transformation est représentée par une ontologie dite de transformation constituée de quatre composantes :

- *une ontologie du langage semi-formel* représente les éléments du vocabulaire et de la grammaire du langage semi-formel utilisé pour construire les modèles ;
- *une ontologie de traitement des ambiguïtés et des erreurs* contient des règles servant à la désambiguïsation et à l'identification des erreurs de catégorisation ;
- *une ontologie de référence* représente les diverses catégories de connaissances et opérationnalise le processus de désambiguïsation et de transformation ;
- *une ontologie cadre* encadre les éléments du modèle cible. Agissant comme le modèle d'ontologies de l'ontologie à construire, l'ontologie cadre sert à structurer l'ontologie construite.

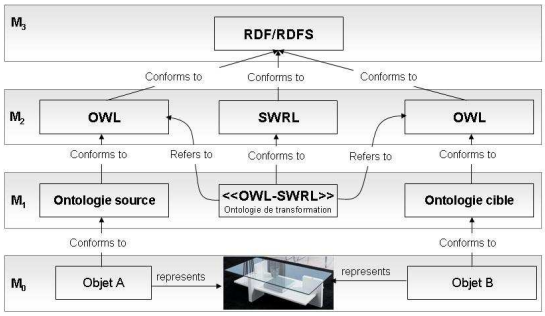

FIGURE 2.11 – Transformation de modèles dans OntoCASE.

2.10 Conclusion

Les transformations de modèles peuvent être utilisées pour différentes tâches tout au long du processus de développement pour la manipulation de modèles. Les transformations de modèles sont ex-

primées dans les langages de transformation de modèles qui peuvent être, soit des grammaires ou des méta-modèles (approche déclarative), soit un des programmes (approche impérative). Ce chapitre nous a permis, d'une part, de fixer la terminologie se rapportant à la transformation de modèles telle que nous l'utilisons dans cette thèse, et, d'autre part, de présenter quelques scénarios d'utilisation des transformations de modèles. Nous avons d'abord identifié certaines caractéristiques des problèmes qui peuvent être résolus à l'aide de transformations de modèles. Nous avons ensuite synthétisé un système de classification pour les transformations de modèles. Plusieurs langages et outils pour les transformations de modèles ont été développés ces dernières années. Nous avons présenté les plus représentatifs d'entre eux en décrivant leurs caractéristiques. Nous avons également, dans ce chapitre, montré que les bases de données à base de modèles sont mieux adaptées pour la gestion des mappings car non seulement elles passent mieux à l'échelle mais elles disposent également d'un niveau sémantique pour la représentation des modèles. Nous pensons d'ailleurs que ce niveau sémantique peut être enrichi pour servir d'espace pour la représentation à la fois des modèles et des mappings. La réalisation d'une telle approche passe à la fois par la formalisation de la manière de représenter les mappings dans une base de données ainsi qu'à la proposition des mécanismes pour la manipulation explicite des mappings. L'un des objectifs dans cette réalisation est de fournir un système de gestion des mappings qui, pour un utilisateur du système, masque autant que possible la complexité des traitements inhérents à la manipulation des mappings.

Synthèse générale de l'état de l'art

Au regard des outils d'aide à la construction que nous avons présentés, nous pouvons les regrouper en deux principales catégories. D'une part, nous avons des outils pour lesquels la représentation du passage d'une étape à l'autre est figée (Text2Onto et Terminae par exemple), et, d'autre part, des outils offrant plus de souplesse pour la représentation du passage d'une étape à l'autre. Cette dernière catégorie d'outils est généralement conçue par un informaticien, en collaboration avec un expert du domaine de la construction d'ontologies. L'expert propose la démarche ainsi que les règles de passage d'une étape de modélisation à l'autre (de la terminologie au thésaurus par exemple) et l'informaticien modélise et implémente la démarche proposée. Par exemple, c'est l'expert qui décide des similarités existantes entre les constituants d'une terminologie et ceux d'un thésaurus. L'utilisateur final doit donc apprendre les règles de l'expert et si malheureusement il n'est pas d'accord (ce qui peut arriver lorsque l'utilisateur final est lui aussi expert) avec l'une de ces règles, celui-ci est obligé de l'accepter ou alors de ne plus utiliser l'outil. Notre travail s'intéresse donc à la possibilité d'embarquer l'utilisateur final (potentiellement expert) dans la description du mécanisme de passage d'une étape de modélisation à l'autre. Il s'agit pour nous de proposer une approche générique qui, à partir de la description de règles par un utilisateur, est capable de supporter la démarche de construction d'ontologies en garantissant notamment la cohérence des données du processus de construction.

Bien que des outils tels que OntoCASE, Text2Onto ou Terminae proposent des approches progressives de construction d'ontologies, on ne peut cependant pas affirmer que l'un d'eux permet de résoudre la problématique du projet DaFOE4App qui, rappelons le, vise à construire une plate-forme supportant plusieurs méthodes et modèles, et offrant la possibilité d'intégrer les résultats des différents outils actuels. Par exemple, l'aspect - à notre avis intéressant - de description informelle des concepts, tel que proposé par DOE, n'est prise en charge par aucun de ces outils. Par ailleurs, la structure de modélisation figée de ces outils, ainsi que la difficulté de les connecter avec d'autres outils ne répondent pas aux exigences E_2 et E_3 visant respectivement à autoriser la modification dynamique des modèles associés aux différentes étapes du processus de construction d'ontologies et à permettre l'insertion, dans la plate-forme DaFOE, de nouveaux traitements utilisables pour passer d'une étape à une autre.

De plus, comme la plate-forme DaFOE utilise une base de données relationnelle comme support de persistance des données et au regard de la cartographie des approches de transformation de modèles ainsi que des outils support existants, notre préoccupation est double. Il s'agit, d'une part, de choisir l'approche la mieux adaptée aux exigences de la plate-forme DaFOE, et, d'autre part, de choisir un langage de

transformation à utiliser pour la représentation des transformations au sein de la plate-forme DaFOE. Si l'exigence d'évolutivité des modèles au sein de la plate-forme rend trivial le choix de l'approche par méta-modélisation, la question du choix du langage est plus difficile à décider. Cette difficulté est due à deux principaux facteurs :

1. *Stockage uniforme des modèles, des instances et des mappings*.
   Les données de la plate-forme DaFOE devant être stockées dans une base de données relationnelle, l'utilisation d'un des langages présentés précédemment nous imposerait, d'une part, de disposer d'un système externe (à l'environnement de la base de données) pour stocker à la fois les modèles et les transformations, et, d'autre part, de garantir la synchronisation entre le système externe et la base de données ;

2. *Gestion uniforme des modèles, des instances et des mappings*.
   Le terme gestion fait référence ici aux opérations de création, de suppression, de modification et de lecture des modèles, des instances et des mappings. Un stockage non-uniforme aurait comme conséquence immédiate l'apprentissage d'un nouveau langage (celui décrivant les modèles et les mappings) dont la syntaxe et les concepts ne sont pas forcément proches de ceux de la gestion des données.

Bien que l'outil OntoCASE utilise la transformation de modèles, cet outil ne s'intéresse pas à l'évolution du processus de construction d'ontologies. Par évolution du processus, nous entendons la capacité de l'outil à permettre à un utilisateur de supprimer ou d'ajouter une ou plusieurs étape(s) dans le processus de construction d'ontologies. Un utilisateur peut, par exemple, décider de raffiner son processus de construction en créant une étape intermédiaire entre deux étapes. Dans cette approche que nous préconisons, quelques questions sont fondamentales :

- est-il possible de le faire dynamiquement en simplifiant autant que possible, les modifications à apporter sur le processus de construction initial?
- comment garantir la cohérence du système résultant de l'ajout, de la suppression ou de la modification d'une étape?
- comment faciliter l'exploitation du système résultant de l'ajout, de la suppression ou de la modification d'une étape?

Sur le plan de la transformation de modèles, la contribution de cette thèse consiste, d'une part, à proposer, une infrastructure permettant de stocker de manière uniforme les modèles, les instances ainsi que les mappings, et, d'autre part, à proposer un mécanisme facilitant l'exploitation des modèles, des instances et des mappings. Nous montrons d'ailleurs que notre approche est générique du fait de sa capacité à supporter à la fois l'évolution dynamique des modèles et du langage de transformation. Cette gestion de l'évolution nous permet notamment d'utiliser, au sein de la plate-forme DaFOE, une combinaison de plusieurs outils de TAL, et donc, à partir d'un corpus de textes, d'obtenir de nouveaux paramètres d'analyse des termes pertinents pour la construction d'une ontologie. Notons néanmoins que notre travail ne s'intéresse pas à la génération de modèles mais plutôt à la spécification des mappings ainsi que leur exploitation pour la gestion des instances des modèles. De ce fait, le moteur de transformation interprète

les mappings pour, par exemple, construire à la volée, les instances du modèle cible à partir des instances du modèle source.

Par ailleurs, nous avons également vu que les BDBM, grâce à leur architecture de méta-modélisation, étaient mieux adaptées pour la persistance des mappings. Pour cette raison, l'approche offerte par les langages pour les systèmes à base de méta-modélisation mérite d'être envisagée si l'on souhaite manipuler les mappings. Les travaux réalisés pour les langages orientés mapping constituent donc un excellent point de départ pour caractériser un langage de requête exploitant les mappings. Ainsi, du point de vue de l'exploitation des mappings, la contribution de notre thèse consiste donc à capitaliser les avantages de chacune des catégories de langages présentées dans le chapitre 2 (Cf. section 2.8) pour ensuite proposer un langage de requête s'affranchissant des limitations de ces catégories de langages. Il s'agit, dans un premier temps de proposer un ensemble d'exigences caractérisant les langages d'exploitation des mappings, et, dans un second temps, de proposer un langage de requête conforme à ces exigences.

Deuxième partie

Contribution

"Nothing great was ever achieved without enthusiasm."
– Ralph Waldo Emerson

Chapitre 3

Modélisation

Sommaire

Résumé. Dans la première partie de ce mémoire, nous avons d'abord présenté une vue globale des approches de construction d'ontologies, puis nous avons détaillé les approches qui, comme celle que nous présentons dans cette thèse, utilisent des textes comme données d'entrée. Nous avons évoqué que notre approche se distingue des autres à la fois par l'utilisation des transformations de modèles, mais aussi par la manière dont nous modélisons ces transformations. Ces différentes particularités sont présentées dans ce chapitre qui décrit la démarche de construction d'ontologies dans la plate-forme DaFOE ainsi que notre approche pour la représentation des modèles et des transformations nécessaires.

3.1 Introduction

Dans le chapitre 1, nous avons vu que certains outils permettaient de construire des ontologies à partir de textes selon une démarche en étapes sans toutefois proposer une réelle séparation (du point de vue de la modélisation) des étapes impliquées dans le processus de construction. Cette séparation est pourtant fondamentale si l'on souhaite préserver la traçabilité de la démarche de construction d'ontologies et ainsi offrir à l'utilisateur, la possibilité de comprendre et d'influer cette démarche. Aussi, dans notre approche [Charlet et al., 2008, Téguiak et al., 2010], cette séparation est explicitement représentée. En effet, nous modélisons, dans un premier temps, chaque étape indépendamment des autres. Puis, nous définissons un espace pour la description des mappings entre les modèles associés aux différentes étapes. Enfin, un programme générique exploitant ces mappings permet une migration semi-automatique des données d'une étape à l'autre. Les modèles associés aux différentes étapes étant susceptibles d'évoluer, nous utilisons une approche par méta-modélisation qui nous permet de représenter à la fois les modèles et les mappings entre ces modèles. De cette façon, nous montrons que le processus de construction d'ontologies, qui ne comporte que trois étapes dans sa version initiale au sein de plate-forme DaFOE, peut être généralisé à un processus de construction comportant un nombre quelconque d'étapes. La création d'une nouvelle étape consiste à créer le modèle représentant cette étape, et, d'autre part à créer les mappings permettant d'interconnecter des modèles. Dans ce chapitre, nous présentons tout d'abord le cadre méthodologique de construction d'ontologies offert par la plate-forme DaFOE (Cf. section 3.2), puis nous présentons, dans la section 3.3, l'approche à base de transformations de modèles que nous proposons pour la mise en œuvre de cette démarche. Dans les sections 3.4 et 3.5, nous décrivons formellement notre représentation des modèles et des mappings.

3.2 Construction d'ontologies : cas de la plate-forme DaFOE

Initié dans le cadre du projet DaFOE4App, la plate-forme DaFOE (Cf. figure 3.1) a pour but de fournir un environnement complet pour la construction d'ontologies en utilisant diverses méthodes. Toutefois, le scénario principal reste celui de la construction d'ontologies à partir de textes. Ainsi, la plate-forme DaFOE vise à fournir une infrastructure capable de supporter le processus de construction d'ontologies depuis l'acquisition de connaissances linguistiques (extraites par des outils de TAL), l'analyse de ces connaissances, jusqu'à leur formalisation au sein d'une ontologie. Parmi les cadres d'expérimentation de la plate-forme DaFOE, nous pouvons citer les domaines d'application tels que le patrimoine, la médecine, l'imagerie satellitaire, etc..

La conception d'un outil de construction d'ontologies nécessite d'une part, de préciser un cadre méthodologique décrivant la démarche de construction, et, d'autre part, de proposer une modélisation de cette démarche. Dans cette section, nous présentons le cadre méthodologie utilisée dans la plate-forme DaFOE. Ce cadre méthodologique s'aligne d'ailleurs sur la démarche Terminae dont les auteurs sont également partenaires du projet DaFOE4App.

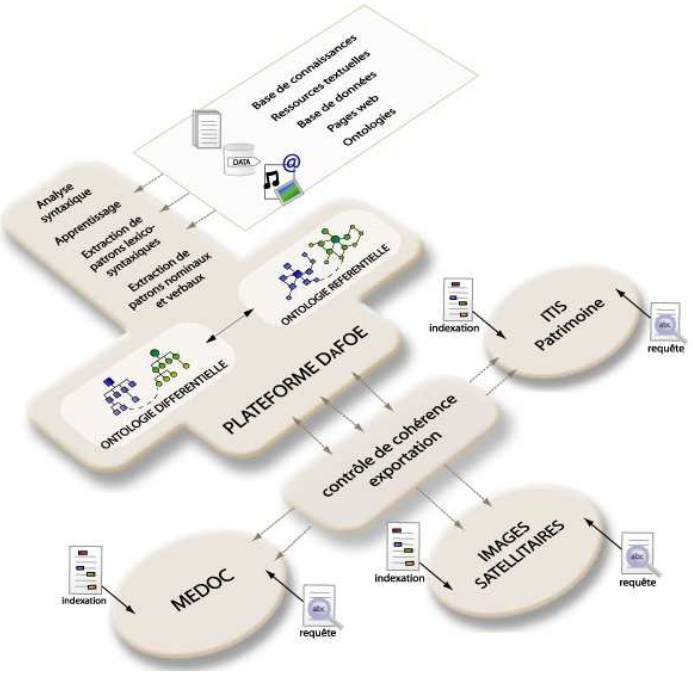

FIGURE 3.1 – Vue globale de la plate-forme DaFOE (Source: `www.dafoe4app.fr`).

3.2.1 Cadre méthodologique

Le processus de construction d'ontologies à partir de textes proposé dans la plate-forme DaFOE comporte trois étapes auxquelles sont respectivement associés les modèles terminologique, termino-ontologique et ontologique. Ce trois étapes consiste respectivement en :

1. extraction des éléments de connaissances linguistiques (termes, relations entre termes, etc.) en utilisant les outils de TAL. Cette étape aboutit à l'instanciation des données du modèle terminologique ;

2. désambiguïsation des éléments de connaissances extraits par les outils de TAL. Cette étape consiste à instancier, de manière semi-automatique, le modèle termino-ontologique à partir des données issues du modèle terminologique ;

3. conception de l'ontologie. Cette étape consiste à instancier, de manière semi-automatique, le modèle ontologique à partir des données du modèle termino-ontologique.

La plate-forme se devant de supporter plusieurs outils de TAL, ces outils ne font pas partie de la plate-forme DaFOE. Ceci simplifie l'architecture globale de la plate-forme qui dispose uniquement d'un programme permettant l'importation de données issues de ces outils.

Dans la démarche utilisée dans la plate-forme DaFOE, les modèles sont associés à différentes étapes de construction d'une ontologie. Nous détaillons, dans cette section, le rôle et la structure de chacune des trois étapes associées à cette démarche.

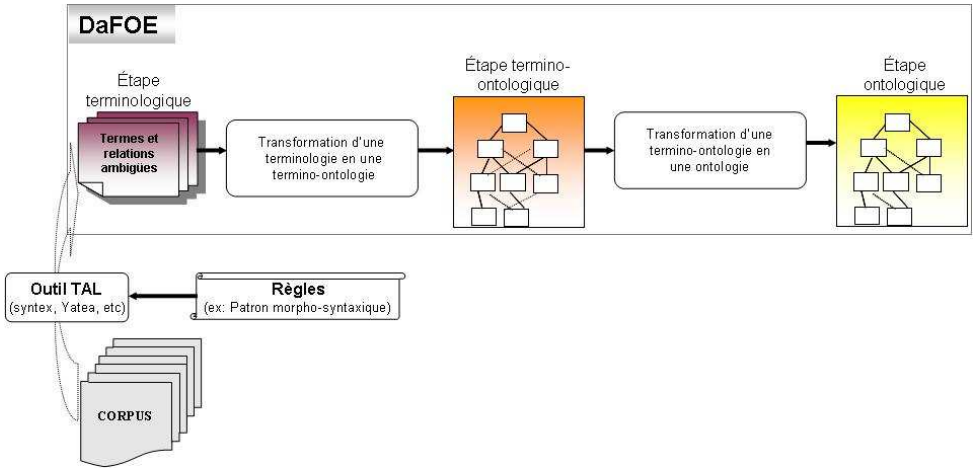

FIGURE 3.2 – Cadre méthodologique de construction d'ontologies dans DaFOE.

3.2.1.1 Étape terminologique

L'étape terminologique sert à représenter les termes et relations extraits par les outils de TAL. Il s'agit de modéliser les résultats bruts issus des outils de TAL. Ces résultant peuvent faire l'objet d'une validation manuelle. Le modèle associé à cette étape permet également de représenter les patrons morpho-syntaxiques ayant permis d'extraire des relations. Les données représentées dans cette étape peuvent être utilisées pour faire une nouvelle conceptualisation à partir du même corpus déjà analysé par les outils de TAL en enrichissant par exemple, ce corpus d'un nouveau document. Dans cette section, nous présentons les principaux éléments manipulés dans l'étape terminologique.

Corpus

Dans la plate-forme DaFOE, les corpus de textes sont modélisés de manière à offrir une visualisation en contexte (dans les documents) des termes utilisés pour construire les concepts de l'ontologie. Un corpus peut contenir plusieurs documents et peut être stocké dans plusieurs fichiers. Le texte brut est découpé en phrases possédant un identifiant unique ainsi qu'un contenu représentant la forme écrite de la phrase.

Termes/Candidats termes

Dans le modèle noyau de la plate-forme DaFOE, les termes sont décrits par :
- un statut pouvant prendre l'une des valeurs *"neutre", "éliminé", "rejeté", "conceptualisé", "en attente"* ;
- des propriétés morpho-syntaxiques (*le genre et le nombre* par exemple) ;
- une forme canonique (éventuellement une variante choisie arbitrairement, à défaut d'autres critères) ;
- des variantes, représentant à la fois les différentes déclinaisons syntaxiques et les synonymes du terme ;

- des occurrences, identifiant les différentes positions d'apparition du terme dans le corpus ;
- des critères de saillance, utilisés pour la comparaison des termes. Par défaut, les critères de saillance sont les suivants : la fréquence, la répartition (par exemple le poids tf.idf ou le facteur idf dans le tf.idf), le nombre de variantes, les productivités (en tête et en expansion), le poids typographique et le poids de position textuelle ;
- un langage, représentant la langue d'usage du terme. Par défaut, la langue du corpus est utilisée.

Dans la plate-forme DaFOE, un candidat terme est représenté et interprété comme un terme en étude, c'est-à-dire un terme dont le statut possède la valeur "en attente". Un *terme conceptualisé* (dont le statut possède la valeur "conceptualisé") est un *terme* pour lequel une donnée, a déjà été créée dans l'étape termino-ontologique.

En guise d'illustration, la figure 3.3 présente le modèle simplifié associé à l'étape terminologique, permettant de représenter des *termes*, des relations entre *termes*, etc.

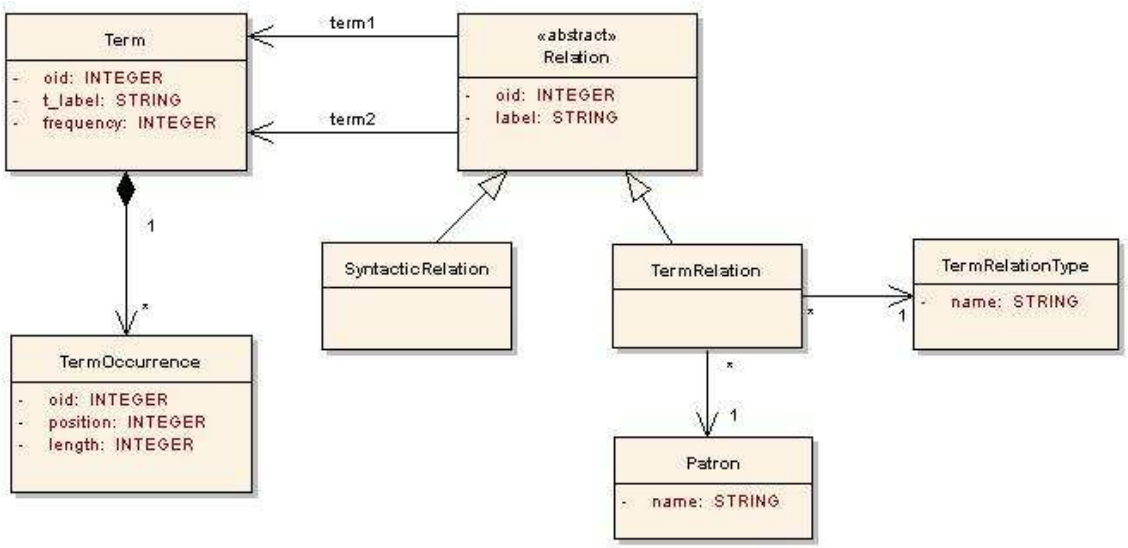

FIGURE 3.3 – Modèle associé à l'étape terminologique.

3.2.1.2 Étape termino-ontologique

Les outils de TAL ne discriminant pas la polysémie des mots, l'étape termino-ontologique sert, d'une part, à désambiguïser le sens des termes, et, d'autre part, grâce aux relations sémantiques, à construire un réseau termino-conceptuel. Elle permet de représenter les termes désambiguïsés (*terminoconcepts*), ainsi que les relations désambiguïsées (*relations terminoconceptuelles*). En cas d'ambiguïté dans l'étape terminologique, tout ou partie des occurrences de l'étape terminologique sont référencées par chacun des *terminoconcepts* résultant du même terme. Cette étape contient tous les résultats intermédiaires et finaux résultant de la désambiguïsation des *termes* et exploite les avantages de l'outil DOE (multilinguisme et principe différentiel). Dans cette étape, un *terminoconcept* est principalement défini par les éléments suivants :

– un terme vedette correspondant, pour l'ensemble des termes associés au *terminoconcept*, à celui le plus représentatif ;
– du texte libre contenant la définition du *terminoconcept* ;
– les quatre critères de différenciation en langage naturel (la *différence* et la *similitude* avec le *terminoconcept père*, la *différence* et la *similitude* avec les *terminoconcepts frères*).

En guise d'illustration, la figure 3.4 présente le modèle simplifié associé à l'étape termino-ontologique. Ce modèle permettent de représenter des *terminoconcepts*, des relations entre *terminoconcepts*, etc.. La classe `TcOccurrence` du modèle termino-ontologique permet de contextualiser le sens des *terminoconcepts* à l'aide de référence vers le sous-ensemble d'occurrences de termes du modèle terminologique dont le sens est identique à celui du *terminoconcept*.

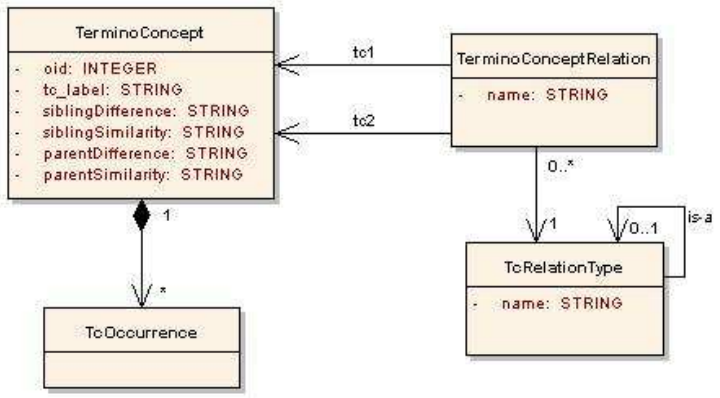

FIGURE 3.4 – Modèle simplifié associé à l'étape termino-ontologique.

3.2.1.3 Étape ontologique

Comme nous l'avons vu dans le chapitre 1 (Cf. section 1.4), une ontologie est mieux adaptée pour la représentation des relations propres à un domaine. Cette représentation se fait grâce à un modèle d'ontologies. Dans la plate-forme DaFOE, le modèle d'ontologie correspond au modèle associé à cette étape. La figure 3.5 présente le modèle simplifié associé cette étape. Ce modèle permet de représenter des ontologies décrites en termes de *classes* et de *propriétés*. Une classe possède une *extension* constituée d'un ensemble d'*instances* encore appelée *individus*, et qui désignent des objets du domaine. Une instance, identifiée par son appartenance à une classe, est décrite par un ensemble de valeurs de propriétés.

3.2.2 Illustration

Dans cette section, nous présentons, suivant la méthodologie proposée dans la plate-forme DaFOE, un exemple de construction d'ontologies à partir d'un corpus de textes issu du domaine du patrimoine.

1. Pour le domaine du patrimoine, l'analyse terminologique consiste au recensement des termes d'un corpus représentatif de ce domaine. Le tableau 3.1 présente un extrait de corpus. En guise d'illus-

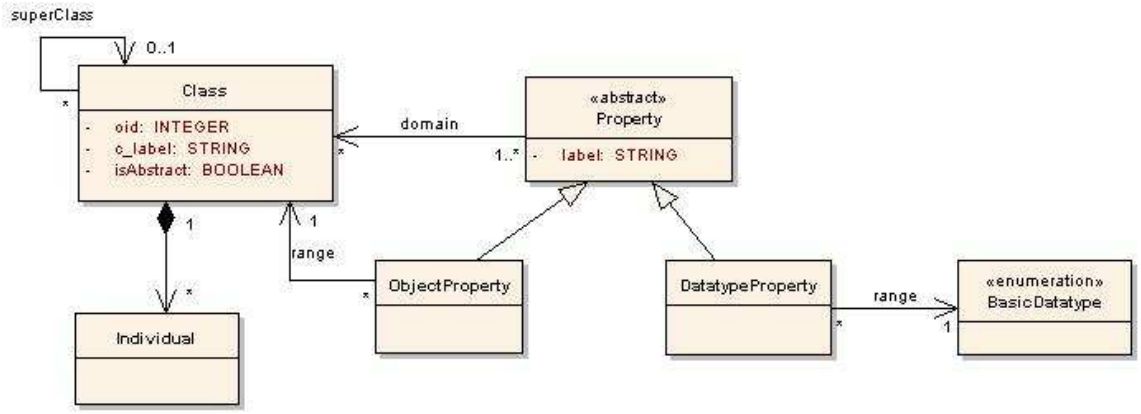

FIGURE 3.5 – Modèle associé à l'étape ontologique.

tration du processus de construction, nous avons marqué en gras les mots pertinents pour le domaine d'étude. Ce corpus est donc, dans un premier temps, traité par un outil de TAL qui produit les éléments terminologiques dont un certain nombre sont présentés dans le tableau 3.2 dans lequel les termes représentant des noms propres (on parle d'entité nommée) se distinguent de ceux représentant des noms communs.

A l'aide de patrons morpho-syntaxiques, un outil de TAL, appliqué au corpus du patrimoine, permet d'extraire les relations présentées dans le tableau 3.3.

2. Dans l'étape d'analyse termino-ontologique, la polysémie des termes est utilisée afin de discriminer, par exemple, le terme "union" qui, dans un contexte désignerait un "mariage" et dans un autre contexte correspondrait à un opérateur algébrique.
   Cette étape permet également, pour les termes "Homme" et "homme" dont la forme canonique commune produite par l'outil de TAL est "homme", d'identifier le premier comme désignant un être humain c'est-à dire une personne indépendamment de son sexe, tandis que le second terme désigne une personne de sexe masculin.

3. Dans l'étape d'analyse ontologique, les données issues de l'analyse termino-ontologique sont organisées en *classes* et *propriétés* pour construire l'ontologie présentée dans la figure 3.6. On remarque par exemple que les *terminoconcepts Personne, Homme, Femme, Mariage, etc.* de l'étape termino-ontologique deviennent des *classes* dans l'étape ontologique tandis que les *relations terminoconceptuelles* (instance de l'entité `TerminoConceptRelation` de la figure 3.4) *enfant, marié, mariée, etc.* deviennent des *propriétés* (instance de l'entité `ObjectProperty` de la figure 3.5).

Tout au long de cette section, nous avons présenté une vue globale du cadre méthodologie proposée par la plate-forme DaFOE pour la construction d'ontologies. Dans les sections suivantes, nous présentons notre contribution pour la mise en œuvre de cette démarche. Nous proposons notamment de modéliser les étapes du processus de construction d'ontologies en associant, à chaque étape, un modèle permettant

Tableau 3.1 – Corpus du domaine du patrimoine

[...]. Le **mariage** est défini traditionnellement comme l'union légitime d'un **homme** et d'une **femme**. Il est l'acte officiel et solennel qui institue entre deux **époux** une communauté de patrimoine et de renommée appelée *famille* dont le but est de constituer de façon durable un cadre de vie commun aux **parents** et aux *enfants* pour leur éducation. [...] Dans les pays où les institutions politiques sont séparées des institutions religieuses, on distingue le **mariage civil** du **mariage religieux** (lequel requiert généralement un mariage civil au préalable). [...] Le **mariage** est aussi, selon les époques et les lieux, une manière d'établir des alliances entre tribus ou familles, une manière de transmettre des biens, une manière de sceller une alliance ou la paix, de réclamer une position de pouvoir, d'obtenir un capital (la dot). Plus connue sous les appellations de **mariage traditionnel** ou de **mariage coutumier**, cette célébration est en générale un préalable pour le **mariage civil** et le **mariage religieux**. [...] Dans certains cas, les **époux** ne peuvent contracter un nouveau mariage tant que le premier est valide ; on parle alors de **monogamie**. Ce **type d'union**, prescrit depuis l'Antiquité, tant par la tradition gauloise, que par la tradition chrétienne, se retrouve dans le droit positif de tous les pays européens. En d'autres temps et d'autres lieux, des mariages peuvent être contractés simultanément avec plusieurs personnes en même temps ; on parle alors de **polygamie**. [...] Afin d'apporter une certaine caution à la sincérité du mariage (tant civil que religieux), les futurs époux doivent faire appel à des **témoins** qui en garantissent la validité. [...] Le droit français, par exemple, laisse la liberté aux époux de choisir leur **régime matrimonial** en rédigeant un **contrat de mariage** au moment du mariage ou, éventuellement, pendant la vie commune.[...] Bien que le mari soit considéré comme le chef de famille, la vie dans un foyer ne doit pas enfreindre la Déclaration Universelle des Droits de l'**Homme** adoptée le 10 décembre 1948 à **Paris** par l'Assemblée Générale des **Nations Unies**. Cette Déclaration stipule d'ailleurs, dans son article premier, que tous les **êtres humains** naissent libres et égaux en dignité et en droits. [...]

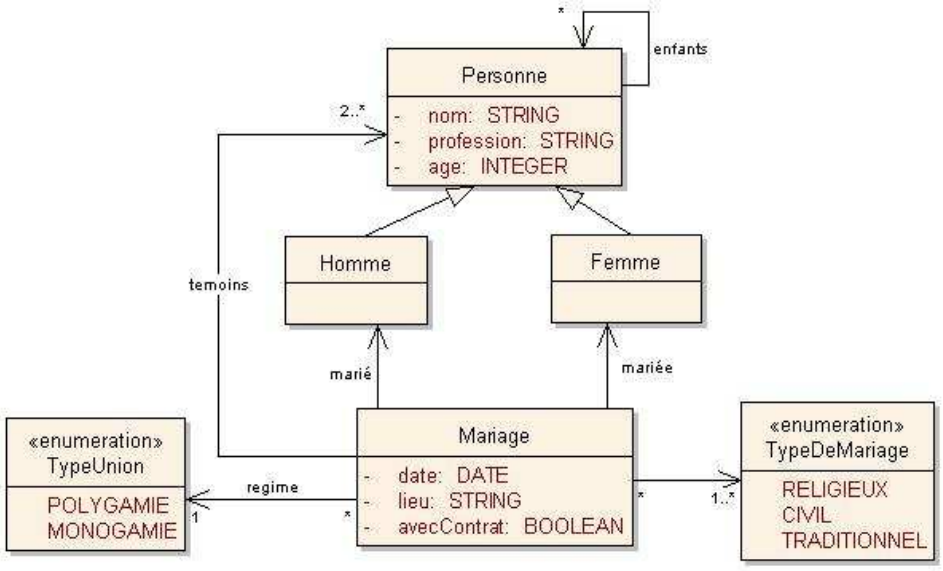

FIGURE 3.6 – Extrait de l'ontologie créée à partir du corpus du patrimoine.

Tableau 3.2 – Quelques termes du corpus du domaine du patrimoine.

Termes (vedette)	Synonymes	estEntitéNommée	statut	...
civil		non	...	...
contrat de mariage		non	...	...
femme		non	...	...
homme		non	...	...
mariage	union	non	...	...
mariage civil		non	...	...
mariage coutumier		non	...	...
mariage religieux		non	...	...
mariage traditionnel		non	...	...
monogamie		non	...	...
Nations Unies		oui	...	...
Paris		oui		
Personne	Homme, être humain	non	...	...
polygamie		non	...	...
religieux		non		
régime matrimonial		non	...	...
traditionnel		non	...	...
type d'union		non	...	...
...	...	...	...	...

Tableau 3.3 – Quelques relations du corpus du domaine du patrimoine.

Relations	Synonymes	Patrons morpho-syntaxiques	statut	...
conjoint		est XXX de	validé	...
enfant		est XXX de	validé	...
marié	époux	est XXX à	validé	...
mariée	épouse	est XXX à	validé	...
parent		est XXX de	validé	...
témoin		est XXX de	validé	...
...	...	...	...	...

de stocker les données manipulées dans cette étape. Cette approche qui vise à rendre les étapes quasi-autonomes les unes des autres, nous impose, d'une part, de pouvoir manipuler les différents modèles associés aux étapes, et, d'autre part, de pouvoir exprimer et manipuler le passage d'une étape à une autre. La démarche que nous adoptons pour la modélisation de ce processus est similaire à celle utilisée pour les systèmes à base de transformations dans lesquels des transformations représentent des similarités conceptuelles entre des modèles et peuvent être utilisés, par exemple, pour exprimer les règles applicables lors de l'importation des instances d'un modèle à l'autre.

3.3 Notre Approche

Pour les besoins de la plate-forme DaFOE, l'approche que nous proposons dans cette thèse doit fournir des mécanismes permettant l'évolution des trois modèles associés aux étapes présentées précédemment. En effet, comme nous l'avons précisé dans le chapitre 1 (Cf. section 1.5.2.2), plusieurs outils

de TAL pouvant être utilisés, il est nécessaire de pouvoir adapter le modèle associé à l'étape terminologique en fonction de l'outil de TAL utilisé. De la même manière, le modèle associé à l'étape termino-ontologique doit pouvoir être adapté en fonction des données à importer (thésaurus par exemple). Ce besoin d'évolution/adaptation de modèles se fait également ressentir dans l'étape ontologique lorsqu'on souhaite construire une ontologie à partir d'une ontologie existante et représentée dans un modèle d'ontologie différent du modèle associé à l'étape ontologique dans la plate-forme DaFOE.

Par ailleurs, le processus de construction d'ontologies proposé dans la plate-forme DaFOE se faisant par étape, les modèles associés à ces étapes doivent être connectés pour garantir la traçabilité du processus de construction.

Compte tenu de ces considérations, notre approche consiste à :

1. *rendre autonome, autant que possible, les modèles associés aux différentes étapes*. Cette autonomie des modèles vise à minimiser les répercussions de leur éventuelle évolution sur les autres étapes. En effet, en utilisant une association dite *"codée en dur"* (à l'aide de référence par des clés étrangères, par exemple), la suppression d'une entité (l'entité `Term` par exemple) ou d'un attribut du modèle terminologique pourrait entraîner la modification des modèles liés au modèle terminologique. Cette modification pourrait, par exemple, se traduire par la suppression de l'entité `TerminoConcept` du modèle termino-ontologique qui, à son tour, pourrait nécessiter la suppression en cascade des entités, dans le même modèle termino-ontologique, et pour lesquelles il existe une relation de dépendance matérialisée par une clé étrangère.

2. *représenter la liaison entre deux étapes par des mappings*. Ces mappings sont ensuite utilisés pour semi-automatiser le processus de construction car ils décrivent comment construire les données d'un modèle à partir des données d'un autre modèle.

3. *proposer un méta-modèle permettant de représenter les modèles*. En effet, pour décrire les mappings entre deux modèles, nous devons au préalable pouvoir représenter ces modèles. De plus, ce méta-modèle fournit également les opérateurs permettant de faire évoluer les modèles.

Nous venons de voir que la modification des modèles n'est pas sans conséquence sur la cohérence du mappings. C'est aussi pour cette raison que nous proposons de représenter les mappings au sein d'un modèle (appelé modèle de mapping) de manière à contrôler leurs cohérences. Notre démarche est résumée sur la figure 3.7, où, d'une part, nous utilisons un méta-modèle permettant de créer, de modifier ou de supprimer les modèles associés aux différentes étapes du processus de construction d'ontologies, et, d'autre part, nous utilisons un modèle de mapping pour décrire les mappings entre les modèles. Comme nous le verrons dans la section 3.5, les composantes du modèle de mapping possèdent des associations avec les composantes du méta-modèle. Par conséquent le méta-modèle et le modèle de mapping appartiennent au même niveau d'abstraction. Par la suite, nous utiliserons le terme *"méta-modèle noyau"* ou *"Core Metamodel"* pour désigner l'univers constitué à la fois du méta-modèle et du modèle de mapping.

Si le méta-modèle permet de créer des modèles instanciables, le modèle de mappings permet de créer des mappings qui ne seront pas instanciés. Pour cette raison, nous avons préféré, dans cette thèse, d'utiliser le terme *modèle de mappings* plutôt que *méta-modèle de mappings*.

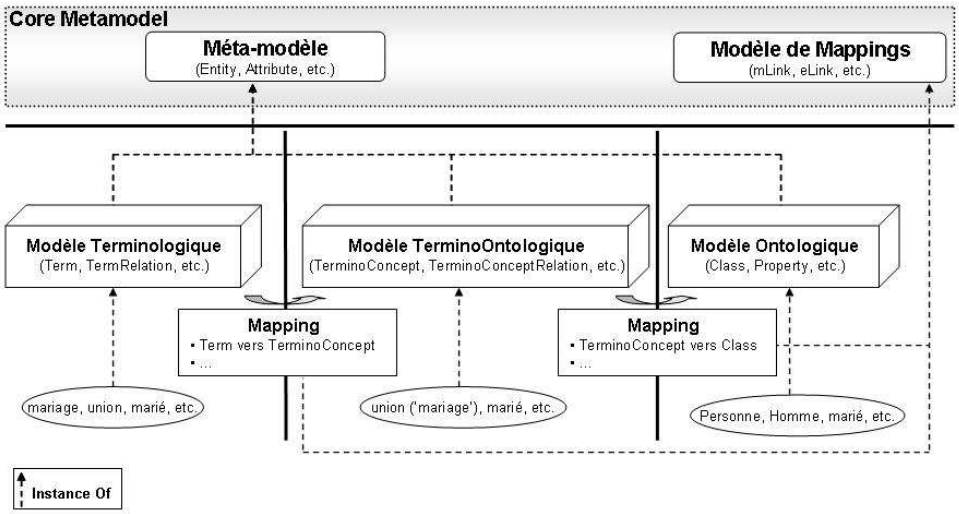

FIGURE 3.7 – Processus de construction à base de mappings.

3.4 Évolution des modèles

Nous avons vu que, dans le cadre de la plate-forme DaFOE, les modèles sont susceptibles d'être modifiés et par conséquent il est important de pouvoir les représenter au sein d'un méta-modèle. Dans cette section, nous présentons le méta-modèle Entité-Attribut que nous avons défini pour représenter les modèles de la plate-forme DaFOE. Auparavant, nous illustrons la notion d'évolution de modèles qui englobe la notion de modification de modèles telle qu'effectuée dans nos travaux.

3.4.1 Illustration

Dans les applications informatiques, les modèles peuvent être représentés grâce à des langages de modélisation tel qu'UML. Si l'on considère le diagramme de la figure 3.8, on y remarque l'évolution d'un modèle d'un état S_i où la *documentation* associée à un *concept* était absente, à un état S_j dans lequel l'utilisateur a décidé qu'une *documentation* d'un *concept* est nécessaire. On peut d'ailleurs, comme l'illustre la figure 3.8, réitérer des modifications sur le modèle de données au moyen de l'opérateur `add` permettant d'ajouter un attribut au modèle ; ou encore en utilisant l'opérateur `delete` permettant de supprimer un attribut du modèle. L'opérateur `rename` permet de renommer un attribut d'une entité. De la même manière, il est possible d'ajouter, de supprimer ou de renommer une classe d'un modèle. Toutes ces opérations modifient un modèle source et produisent un modèle cible.

3.4.2 Méta-modèle Entité-Attribut

Pour supporter l'évolution des modèles, nous définissons ces modèles conformément à un méta-modèle (appelé *méta-modèle Entité-Attribut*) constitué, d'une part, d'entités représentant les concepts du domaine à modéliser, et, d'autre part, d'attributs permettant de décrire les instances de ces concepts. Les modèles que nous manipulons sont conformes aux schémas entités-association de Chen [Chen, 1976] ou

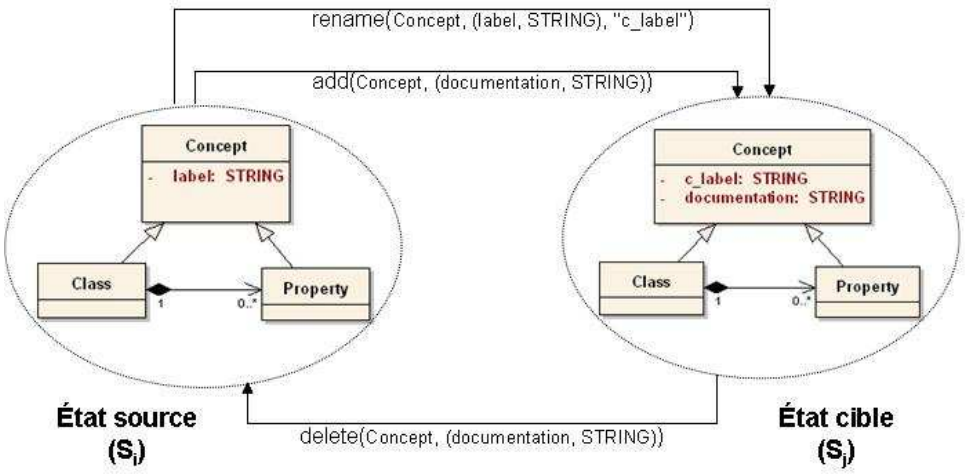

FIGURE 3.8 – Exemple d'évolution des modèles.

aux schémas des bases de données relationnelles. Par conséquent, il n'existe pas de notion d'héritage. Ainsi, tout utilisateur disposant d'une modélisation objet (et donc incluant l'héritage entre concepts du modèle), doit au préalable effectuer une transformation objet/relationnel. Moyennant cette hypothèse, dans le méta-modèle Entité-Attribut, un modèle est représenté de la manière suivante :

Définition 3.4.1

Caractérisation de l'ensemble des modèles

Considérons les ensembles et les applications suivants :

- $\mathcal{M}$ *est l'ensemble des modèles ;*
- $\mathcal{E}$ *est l'ensemble des entités des modèles ;*
- $\mathcal{A}$ *représente l'ensemble des attributs utilisés pour décrire les entités des modèles ;*
- $\mathcal{T}$ *est l'ensemble des types primitifs (Integer, String, Boolean, etc.) utilisés pour typer les attributs des entités ;*
- $\mathcal{I}$ *représente l'ensemble des instances des entités de tous les modèles ;*

- *dom* : $\mathcal{A} \rightarrow \mathcal{E}$ *définit le domaine d'un attribut ;*
- *range* : $\mathcal{A} \rightarrow \mathcal{E} \cup \mathcal{T}$ *définit le codomaine d'un attribut ;*
- *its_model* : $\mathcal{E} \rightarrow \mathcal{M}$ *retourne le contexte de l'entité, c'est-à-dire l'unique modèle associé à l'entité ;*
- *its_entity* : $\mathcal{I} \rightarrow \mathcal{E}$ *retourne le contexte de l'instance, c'est-à-dire l'unique entité associée à l'instance. A l'inverse,* $\forall(\texttt{i},\texttt{e}) \in \mathcal{I} \times \mathcal{E}$ *nous utilisons la notation "*$\texttt{i} \in \texttt{e}$*" pour exprimer que* `i` *est une instance de* `e`.

En considérant les différents modèles de la plate-forme DaFOE (cf. figures 3.3, 3.4 et 3.5), on a, par exemple :

❶ $\mathcal{M}$ = `{Terminology_Model, TerminoOntology_Model, Ontology_Model}`

❷ $\mathcal{E}$ = {Term, TermRelation, TerminoConcept, Class, Property, ...}

❸ $\mathcal{A}$ = {term_label, tc_label, c_label, isAbstract, label, superClass, ...}

❹ $\mathcal{T}$ = {INTEGER, STRING, BOOLEAN, ...}

❺ dom(c_label)= Class

❻ range(superClass) = Class

❼ its_model(Class) = Ontology_Model

❼ etc.

Définition 3.4.2

Caractérisation d'un modèle

Un modèle $\mathtt{m} = (E, A, T, I, dom, range, its\_model, its\_entity) \in \mathcal{M}$ *est défini par :*
- $E \subseteq \mathcal{E}$ *est l'ensemble des entités du modèle* m *i.e* $\mathtt{E} = \{\mathtt{e} \in \mathcal{E} \mid \mathtt{its\_model(e)} = \mathtt{m}\}$;
- $A \subseteq \mathcal{A}$ *représente l'ensemble des attributs utilisés pour décrire les entités du modèle* m *i.e* $\mathtt{A} = \{\mathtt{a} \in \mathcal{A} \mid \mathtt{its\_model(dom(a))} = \mathtt{m}\}$;
- $T \subseteq \mathcal{T}$ *est l'ensemble des types primitifs utilisés pour typer les attributs des entités du modèle* m ;
- $I \subseteq \mathcal{I}$ *représente l'ensemble des instances des entités du modèle* m *i.e* $\mathtt{I} = \{\mathtt{i} \in \mathcal{I} \mid \mathtt{its\_model(its\_entity(i))} = \mathtt{m}\}$;
- $dom : A \rightarrow E$;
- $range : A \rightarrow E \cup T$;
- $its\_entity : I \rightarrow E$.

En guise d'illustration, considérons le modèle Ontology_Model associé à l'étape ontologique (cf. figure 3.5). En utilisant la définition précédente, ce modèle peut être représenté de la manière suivante :

❶ E = {Class, Property, Individual, ObjectProperty, DatatypeProperty, ...}

❷ A = {oid, c_label, isAbstract, label, superClass, ...}

❸ T = {INTEGER, STRING, BOOLEAN}

❹ dom(c_label)= Class;

❺ range(superClass) = Class

❻ its_model(Class) = Ontology_Model

❼ etc.

Notation 3.4.2.1

Accès aux paramètres d'un modèle

Dans ce mémoire, nous manipulons à la fois les structures de données (un modèle par exemple) et le contenu des structures (une instance d'un modèle par exemple). Pour cela, nous utilisons les n-uplets pour la représentation formelle. Considérons, par exemple, la structure $\mathtt{s} = (\mathtt{p}_1, \mathtt{p}_2, ..., \mathtt{p}_n)$, *où*

les p_i *(*$1 \leq i \leq n$*) représentent des variables décrivant la structure* s*. Un contenu* $c = (v_1, v_2, ..., v_n)$ *où les* v_i *(*$1 \leq i \leq n$*) représentent des constantes correspondant aux valeurs affectées aux variables* p_i*. Pour le contenu* c*, la notation utilisée pour accéder à la valeur prise par la variable* p_i *est* $c[p_i]$*. En d'autres termes, le contenu d'une structure de données est interprété comme un tableau indexé par les variables de la structure de données. Par exemple, si* $m_1 = (E_1, A_1, ...)$ *est un modèle, l'expression* $m_1[\mathcal{E}]$ *retourne les entités du modèle* m_1*, c'est-à-dire l'ensemble* E_1*.*

Nous sommes désormais capables, en utilisant le méta-modèle *Entité-Attribut*, de représenter les modèles associés aux différentes étapes de la construction d'une ontologie au sein de la plate-forme DaFOE. Étant donné que chaque modèle se veut autonome, nous utilisons les mappings pour définir les équivalences conceptuelles entre ces modèles. Dans la section suivante, nous présentons la représentation des mappings au sein de la plate-forme DaFOE.

3.5 Représentation des mappings

Un mapping permet d'exprimer des équivalences conceptuelles (ou appariements) entre les composantes des modèles. Dans le cas de la plate-forme DaFOE, nous proposons de représenter ces équivalences par des graphes orientés dont les nœuds sont les instances du méta-méta-modèle et les arcs définissent des appariements. Cette section propose une formalisation de cette notion de graphe d'appariement. Toutefois, notre travail ne s'intéresse pas à la manière d'identifier ou de calculer ces appariements mais à la manière de les représenter pour ensuite les exploiter. [Rahm and Bernstein, 2001, Euzenat and Shvaiko, 2007] fournissent un état de l'art sur les techniques permettant un appariement automatique de modèles. Dans ces différentes techniques de calcul d'appariements, la structure des modèles à apparier est en général utilisée. Cette structure permet, par exemple, de distinguer des appariements au niveau des modèles, des entités, des attributs, etc.. Afin de rester, autant que possible, compatible avec les résultats des travaux sur le calcul automatique d'appariements, nous imposons à notre représentation de mappings de pouvoir supporter ces différents types d'appariements. Ces appariements constituent donc l'ensemble minimal des appariements disponibles dans la plate-forme DaFOE. En effet, comme nous le verrons dans la section 3.6, notre approche offre la possibilité d'étendre la représentation des mappings.

3.5.1 Approche globale

La représentation des mappings au sein de la plate-forme n'a pas été une tâche facile à décider. En effet, si le cadre méthodologique et la représentation des modèles a été relativement rapide à adopter par le consortium du projet DaFOE4App, la représentation des mappings a nécessité plus de réflexions, les idées n'étant pas suffisamment précises sur les mappings à écrire. Plus concrètement, les partenaires du projet ne savaient pas avec exactitude, par exemple, quelle entité du modèle terminologique sera (uniquement ou pas) appariée à une entité précise du modèle termino-ontologique. Face à cela, nous avons proposé de n'exclure aucune éventualité. En d'autres termes, il est question d'envisager, par exemple, qu'une entité du modèle terminologique puisse être appariée à n'importe quelle entité du modèle termino-ontologique et inversement. Cette multitude de possibilités qui rend le graphe d'appariements complexe, multiplie par conséquent les chemins possibles lors de son exploration. Pour améliorer le parcours de

ce graphe, nous avons proposé d'introduire un critère de parcours. Nous utilisons pour cela des attributs issus de la logique floue (Cf. section 2.6 du chapitre 2) qui se matérialise par l'introduction d'un degré de confiance pour les mappings.

Dans le cadre de plate-forme DaFOE, l'implémentation de ces éléments de logique floue est réalisée par des ensembles flous. Ce choix a essentiellement été motivé par la simplicité de mise en œuvre. En effet, en utilisant les ensembles flous, le calcul du degré de confiance d'une composition de deux degrés de confiance dépend uniquement des deux degrés de confiance considérés. Pour la transformation de modèles, nous proposons d'utiliser les ensembles flous pour identifier, par exemple, pour un modèle $\mathtt{m}$ interrogé, une partie floue correspondant à l'ensemble des modèles interrogeables à partir du modèle $\mathtt{m}$. Dans notre travail, les mappings sont donc exploités pour, à partir d'un modèle $\mathtt{m}$, interroger les modèles mappés à ce modèle. Ainsi, le parcours se faisant sur des mappings consécutifs, nous utilisons l'opérateur flou de conjonction décrit par [Zadeh, 1965] et pour lequel la conjonction de deux degrés de confiance correspond aux minimum des deux degrés de confiance considérés. En d'autres termes, si on considère, par exemple, trois modèles $\mathtt{m}_1$, $\mathtt{m}_2$ et $\mathtt{m}_3$ et si si μ_{12} (respectivement μ_{23}) représente le degré d'appartenance du modèle $\mathtt{m}_2$ (respectivement $\mathtt{m}_3$) par rapport à $\mathtt{m}_1$ (respectivement $\mathtt{m}_2$), le degré d'appartenance μ_{13} du modèle $\mathtt{m}_3$ par rapport au modèle $\mathtt{m}_1$ est $\mu_{13} = \min(\mu_{12}, \mu_{23})$.

La notion de α-coupe (Cf. section 2.6.2) nous permet, par exemple, de restreindre le sous-ensemble de modèles également interrogeables à partir d'un modèle.

3.5.2 Appariement de modèles : notion de mLink

Même s'il est possible d'établir un appariement conjoint entre plus de deux modèles, les besoins de la plate-forme DaFOE ne nécessitent que deux modèles : un modèle source et un modèle cible. Dans cette section, nous présentons la formalisation de ce type d'appariement dans le contexte de deux modèles ainsi que les propriétés associées à cette formalisation.

3.5.2.1 Formalisation

Les appariements de modèles sont représentés par un graphe orienté dont les nœuds sont des modèles et les arcs définissent les appariements.

De manière formelle, si $\mathcal{M}$ désigne l'ensemble des modèles, le graphe $\mathcal{G}_\mathtt{m}$ d'appariements de modèles (Cf. figure 3.9) est défini par $\mathcal{G}_\mathtt{m} = (\mathcal{N}_\mathtt{m}, \mathcal{L}_\mathtt{m})$, où :

- $\mathcal{N}_\mathtt{m} \subseteq \mathcal{M}$ représente l'ensemble des nœuds du graphe d'appariements de modèles ;
- $\mathcal{L}_\mathtt{m} = \{(\mathtt{m}_\mathtt{s}, \mathtt{m}_\mathtt{t}, \alpha_\mathtt{m}) \in \mathcal{N}_\mathtt{m} \times \mathcal{N}_\mathtt{m} \times [\mathtt{0}, 1]\}$ représente l'ensemble des appariements entre un modèle source ($\mathtt{m}_\mathtt{s}$) et un modèle cible ($\mathtt{m}_\mathtt{t}$), et $\alpha_\mathtt{m}$, arbitrairement fixé par l'utilisateur, représente le degré de confiance de l'appariement.

En considérant le processus de construction d'ontologies offert par la plate-forme DaFOE, on a, par exemple :

- $\mathcal{N}_\mathtt{m}$ ={`Terminology_Model (TM)`,`TerminoOntology_Model (TOM)`,`Ontology_Model (OM)`}
- $\mathcal{L}_\mathtt{m} = \{\mathtt{mL}_1 = (\mathtt{TM}, \mathtt{TOM}, \mathtt{0.9}), \mathtt{mL}_2 = (\mathtt{TOM}, \mathtt{OM}, 1), ...\}$

Le graphe d'appariements de modèles n'est pas forcément acyclique de manière à permettre à un utilisateur de définir un appariement inverse. Toutefois, du fait de la non-acyclicité de ce graphe, et pour éviter une éventuelle exploration infinie lors de son exploitation, le graphe d'appariements de modèles est parcouru en largeur avec marquage des nœuds déjà visités.

Tout au long de ce mémoire, nous utilisons le terme ***mLink*** pour désigner un appariement entre deux modèles.

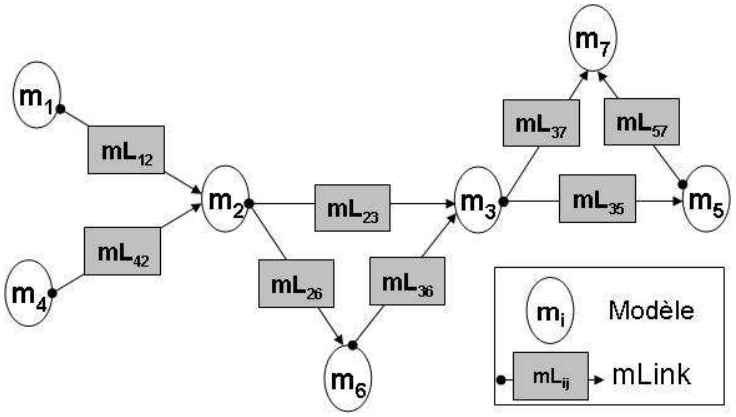

FIGURE 3.9 – Graphe des *mLinks*.

3.5.2.2 Propriétés d'un mLink

L'appariement de modèles permet, lors de l'interrogation des modèles par exemple, d'exploiter les modèles appariés. Les modèles susceptibles d'être utilisés lors d'une telle interrogation correspondent à ceux pour lesquels il est possible de trouver un *mLink* (directement ou par composition de *mLinks*) les reliant. L'opération de composition des *mLinks* n'est possible que pour des *mLinks* consécutifs. Nous parlons alors de *compatibilité des mLinks*. Cette compatibilité est définie de la manière suivante.

Définition 3.5.1

Compatibilité des mLinks

Soient $\mathtt{mL}_1 = (\mathtt{m}_1, \mathtt{m}_2, \alpha_1)$ *et* $\mathtt{mL}_2 = (\mathtt{m}_3, \mathtt{m}_4, \alpha_2)$ *deux mLinks. En s'appuyant sur la définition proposée dans [Nash, 2005],* $\mathtt{mL}_1$ *est dit compatible avec* $\mathtt{mL}_2$ *si* $\mathtt{m}_2 = \mathtt{m}_3$ *et si* $\mathtt{m}_1$, $\mathtt{m}_2$ *et* $\mathtt{m}_4$ *sont deux à deux disjoints. En considérant le graphe de la figure 3.9, on dit, par exemple, que* $\mathtt{mL}_{12}$ *est à la fois compatible avec* $\mathtt{mL}_{23}$ *et* $\mathtt{mL}_{26}$*. Par contre* $\mathtt{mL}_{12}$ *n'est pas compatible avec* $\mathtt{mL}_{42}$*.*

La figure 3.9 montre que les appariements entre les modèles peuvent être exploités de manière transitive de proche en proche. En effet, en considérant les *mLinks* $\mathtt{mL}_{12} = (\mathtt{m}_1, \mathtt{m}_2, \alpha_{12})$ et $\mathtt{mL}_{23} = (\mathtt{m}_2, \mathtt{m}_3, \alpha_{23})$ exprimant l'appariement du modèle $\mathtt{m}_1$ vers le modèle $\mathtt{m}_2$ (respectivement du modèle $\mathtt{m}_2$ avec le modèle $\mathtt{m}_3$), on souhaite caractériser le *mLink* $\mathtt{mL}_{13} = \mathtt{mL}_{12} \circ \mathtt{mL}_{23} = (\mathtt{m}_{s_{13}}, \mathtt{m}_{t_{13}}, \alpha_{13})$ exprimant l'apparie-

ment composé du modèle $\mathtt{m}_1$ avec le modèle $\mathtt{m}_3$. Nous utilisons pour cela l'opérateur de composition des *mLinks* défini par :

$$\begin{array}{rccl} \circ : & \mathcal{L}_\mathtt{m} \times \mathcal{L}_\mathtt{m} & \longrightarrow & \mathcal{L}_\mathtt{m} \\ & (\mathtt{mL}_{12}, \mathtt{mL}_{23}) & \longmapsto & \mathtt{mL}_{13} = (\mathtt{m}_{\mathtt{s}_{13}}, \mathtt{m}_{\mathtt{t}_{13}}, \alpha_{13}) \end{array} \text{ tel que :}$$

$$\begin{cases} \bullet\ \mathtt{m}_{\mathtt{s}_{13}} = \mathtt{mL}_{12}\,[\mathtt{m}_\mathtt{s}] & \text{/* le modèle source du } 1^{er} \text{ mLink*/} \\ \bullet\ \mathtt{m}_{\mathtt{t}_{13}} = \mathtt{mL}_{23}\,[\mathtt{m}_\mathtt{t}] & \text{/* le modèle cible du } 2^{nd} \text{ mLink*/} \\ \bullet\ \alpha_{13} = \min(\mathtt{mL}_{12}\,[\alpha_\mathtt{m}], \mathtt{mL}_{23}\,[\alpha_\mathtt{m}]) & \text{/* le minimum des degrés de confiance */} \end{cases}$$

La composition n'est définie que pour des *mLinks* compatibles.

Proposition 3.5.2.1

La composition est une opération fermée pour l'appariement de modèles. En d'autres termes, la composition de deux mLinks est également un mLink.

Preuve. Par construction, $\mathtt{mL}_{13} = \mathtt{mL}_{12} \circ \mathtt{mL}_{23}$ et on a :
- $\mathtt{m}_{\mathtt{s}_{13}} = \mathtt{mL}_{12}\,[\mathtt{m}_\mathtt{s}] = \mathtt{m}_1 \in \mathcal{N}_\mathtt{m}$
- $\mathtt{m}_{\mathtt{t}_{13}} = \mathtt{mL}_{23}\,[\mathtt{m}_\mathtt{t}] = \mathtt{m}_3 \in \mathcal{N}_\mathtt{m}$
- $\alpha_{13} = \min(\mathtt{mL}_{12}\,[\alpha_\mathtt{m}], \mathtt{mL}_{23}\,[\alpha_\mathtt{m}]) = \min(\alpha_{12}, \alpha_{23}) \leq 1$ car $\mathtt{0} \leq \alpha_{13}, \alpha_{23} \leq 1$

■

Par conséquent, tout au long de ce mémoire, la notion de *mLink* fait à la fois référence aux *mLinks* directs entre deux modèles et aux *mLinks* obtenus par composition de *mLinks*. Notons que cette façon de calculer le degré de confiance du *mLink* résultant de la composition de deux *mLinks* constitue l'approche par défaut au sein de la plate-forme DaFOE. Nous verrons d'ailleurs qu'un utilisateur peut modifier ce mode de calcul à condition que la composition de deux *mLinks* demeure une opération fermée.

3.5.3 Appariement d'entités : notion d'eLink

L'investigation du besoin d'appariement d'entités au sein de la plate-forme DaFOE nous a permis de constater qu'exactement deux entités participent à la fois à ce type d'appariement : une entité source et une entité cible. Dans cette section, nous présentons la formalisation de ce type d'appariement dans le contexte de deux entités ainsi que les propriétés associées à cette formalisation.

3.5.3.1 Formalisation

Les appariements d'entités de deux modèles (tous deux appariés par un *mLink*) sont représentés par un graphe orienté dont les nœuds sont des entités et les arcs définissent les appariements.

De manière formelle, si $\mathcal{E}$ désigne l'ensemble des entités des modèles, le graphe $\mathcal{G}_\mathtt{e}$ d'appariements d'entités (Cf. figure 3.10) est défini par $\mathcal{G}_\mathtt{e} = (\mathcal{N}_\mathtt{e}, \mathcal{L}_\mathtt{e})$ où :

- $\mathcal{N}_\mathtt{e} \subseteq \mathcal{E}$ représente l'ensemble des nœuds du graphe d'appariement d'entités ;
- $\mathcal{L}_\mathtt{e} = \{(\mathtt{e}_\mathtt{s}, \mathtt{e}_\mathtt{t}, \alpha_\mathtt{e}, \mathtt{mL}) \in \mathcal{N}_\mathtt{e} \times \mathcal{N}_\mathtt{e} \times [\mathtt{0}, 1] \times \mathcal{L}_\mathtt{m}\}$ représente l'ensemble des appariements entre une

entité source ($\mathtt{e_s}$) et une entité cible ($\mathtt{e_t}$) avec un degré de confiance $\alpha_\mathtt{e}$ arbitrairement fixé par l'utilisateur. Un appariement d'entités est forcément associé à un *mLink* qui représente son *contexte de création*.

En considérant le processus de construction d'ontologies offert par la plate-forme DaFOE, on a, par exemple :

- $\mathcal{N}_\mathtt{e} = \{\mathtt{Term}, \mathtt{TerminConcept}, \mathtt{Class}, ...\}$
- $\mathcal{L}_\mathtt{e} = \{\mathtt{eL}_1 = (\mathtt{Term}, \mathtt{TerminConcept}, \mathtt{0.8}, \mathtt{mL}_1), \mathtt{eL}_2 = (\mathtt{TerminConcept}, \mathtt{Class}, \mathtt{0.9}, \mathtt{mL}_2), ...\}$

Le graphe d'appariements d'entités n'est pas forcément acyclique de manière à permettre à un utilisateur de définir un appariement inverse. Toutefois, du fait de la non-acyclicité de ce graphe, et pour éviter une éventuelle exploration infinie lors de son exploitation, le graphe d'appariement d'entités est parcouru en largeur avec marquage des nœuds déjà visités.

Tout au long de ce mémoire, nous utilisons le terme ***eLink*** pour désigner un appariement de deux entités.

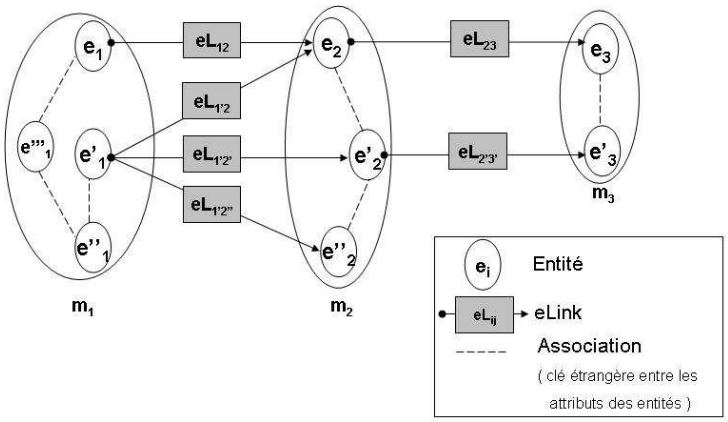

FIGURE 3.10 – Graphe des *eLinks*.

3.5.3.2 Propriétés d'un eLink

L'appariement d'entités permet par exemple, d'exploiter les entités appariées lors de l'interrogation des entités. Les entités susceptibles d'être utilisées lors d'une telle interrogation correspondent à celles pour lesquelles il est possible de trouver un *eLink* (directement ou par composition de *eLinks*) les reliant. La composition de deux *eLinks* n'est possible que pour des *eLinks* consécutifs. Nous parlons alors de *compatibilité des eLinks*.

Définition 3.5.2

Compatibilité des eLinks

Soient $\mathtt{eL}_1 = (\mathtt{e}_1, \mathtt{e}_2, \alpha_1, \mathtt{mL}_1)$ *et* $\mathtt{eL}_2 = (\mathtt{e}_3, \mathtt{e}_4, \alpha_2, \mathtt{mL}_2)$ *deux eLinks créés dans le même contexte*

(i.e $\mathtt{mL}_1 = \mathtt{mL}_2$). De façon analogue à la définition de la compatibilité des mLinks, on dit que $\mathtt{eL}_1$ est compatible avec $\mathtt{eL}_2$ si $\mathtt{e}_2 = \mathtt{e}_3$ et si $\mathtt{e}_1$, $\mathtt{e}_2$ et $\mathtt{e}_4$ sont deux à deux disjoints. En considérant le graphe de la figure 3.10, on peut dire que $\mathtt{eL}_{12}$ est compatible avec $\mathtt{eL}_{23}$ mais n'est pas compatible avec $\mathtt{eL}_{1'2}$.

Comme illustré par le graphe de la figure 3.10, les appariements d'entités peuvent être exploités de manière transitive de proche en proche. En effet, en considérant les *eLinks* $\mathtt{eL}_{12} = (\mathtt{e}_1, \mathtt{e}_2, \alpha_{12}, \mathtt{mL}_{12})$ et $\mathtt{eL}_{23} = (\mathtt{e}_2, \mathtt{e}_3, \alpha_{23}, \mathtt{mL}_{23})$ exprimant l'appariement de l'entité $\mathtt{e}_1$ avec l'entité $\mathtt{e}_2$ (respectivement de l'entité $\mathtt{e}_2$ avec l'entité $\mathtt{e}_3$), on souhaite caractériser le *eLink* $\mathtt{eL}_{13} = (\mathtt{e}_{s_{13}}, \mathtt{e}_{t_{13}}, \alpha_{13}, \mathtt{mL}_{13})$ exprimant l'appariement composé de l'entité $\mathtt{e}_1$ avec l'entité $\mathtt{e}_3$. Nous utilisons pour cela la composition des *eLinks* défini par :

$$\begin{array}{rccl} \circ : & \mathcal{L}_\mathtt{e} \times \mathcal{L}_\mathtt{e} & \longrightarrow & \mathcal{L}_\mathtt{e} \\ & (\mathtt{eL}_{12}, \mathtt{eL}_{23}) & \longmapsto & \mathtt{eL}_{13} = (\mathtt{e}_{s_{13}}, \mathtt{e}_{t_{13}}, \alpha_{13}, \mathtt{mL}_{13}) \end{array} \quad \text{tel que :}$$

$$\begin{cases} \bullet\ \mathtt{e}_{s_{13}} = \mathtt{eL}_{12}\,[\mathtt{e}_s] & \text{/* l'entité source du } 1^{er} \text{ eLink*/} \\ \bullet\ \mathtt{e}_{t_{13}} = \mathtt{eL}_{23}\,[\mathtt{e}_t] & \text{/* l'entité cible du } 2^{nd} \text{ eLink*/} \\ \bullet\ \alpha_{13} = \min(\mathtt{eL}_{12}\,[\alpha_\mathtt{e}]\,, \mathtt{eL}_{23}\,[\alpha_\mathtt{e}]) & \text{/* le minimum des degrés de confiance */} \\ \bullet\ \mathtt{mL}_{13} = (\mathtt{eL}_{12}\,[\mathtt{mL}]) \circ (\mathtt{eL}_{23}\,[\mathtt{mL}]) & \text{/* la composition des mLinks */} \end{cases}$$

Tout comme pour les *mLinks*, la composition des *eLinks* n'est définie que pour les *eLinks* compatibles.

Proposition 3.5.3.2

La composition est une opération fermée pour l'appariement d'entités. En d'autres termes, la composition de deux eLinks est également un eLink.

Preuve. Par construction, $\mathtt{eL}_{13} = \mathtt{eL}_{12} \circ \mathtt{eL}_{23}$ et on a :
- $\mathtt{e}_{s_{13}} = \mathtt{eL}_{12}\,[\mathtt{e}_s] = \mathtt{e}_1 \in \mathcal{N}_\mathtt{e}$
- $\mathtt{e}_{t_{13}} = \mathtt{eL}_{23}\,[\mathtt{e}_t] = \mathtt{e}_3 \in \mathcal{N}_\mathtt{e}$
- $\alpha_{13} = \min(\mathtt{eL}_{12}\,[\alpha_\mathtt{e}]\,, \mathtt{eL}_{23}\,[\alpha_\mathtt{e}]) = \min(\alpha_{12}, \alpha_{23}) \leq 1$ car $0 \leq \alpha_{13}, \alpha_{23} \leq 1$
- $\mathtt{mL}_{13} = (\mathtt{eL}_{12}\,[\mathtt{mL}]) \circ (\mathtt{eL}_{23}\,[\mathtt{mL}]) = \mathtt{mL}_{12} \circ \mathtt{mL}_{23} \in \mathcal{L}_\mathtt{m}$ car la composition des *mLinks* est une opération fermée (Cf. Proposition 3.5.2.1).

■

Par conséquent, tout au long de ce mémoire, la notion de *eLink* fait à la fois référence aux *eLinks* directs entre deux entités et aux *eLinks* obtenus par composition de *eLinks*. Notons que cette façon de calculer le degré de confiance du *eLink* résultant de la composition de deux *eLinks* constitue l'approche par défaut fournie par par la plate-forme DaFOE. Nous verrons d'ailleurs qu'un utilisateur peut modifier ce mode de calcul à condition que la composition de deux *eLinks* demeure une opération fermée.

3.5.4 Appariement d'attributs : notion d'aLink

Dans le cas de la plate-forme DaFOE, les appariements à exprimer entre les attributs ne sont pas forcément de type 1-1, c'est-à-dire qu'un attribut source est apparié avec un et un seul attribut cible. En

effet, pour construire un *terminoconcept* à partir d'un *terme*, on a par exemple besoin d'exprimer le fait que le rendement (`rate`) du *terminoconcept* est égal à somme de la fréquence (`frequency`) du *terme* et du nombre de variantes (`variant_number`) du terme (`rate = frequency + variant_number`). Dans l'étape ontologique par exemple, pour construire, une *classe* à partir d'un *terminoconcept*, on a par exemple besoin d'exprimer que la *pertinence* (`relevance`) de la classe correspond à la valeur normalisée du rendement (*rate*) du *terminoconcept* (`relevance = rate/100`). Pour représenter ce type d'appariement, un modèle d'expressions est nécessaire. Dans cette section, nous présentons à la fois la formalisation de ce type d'appariement, les propriétés associées à cette formalisation ainsi que le modèle d'expressions utilisé.

3.5.4.1 Formalisation

Les appariements d'attributs sont représentés par un graphe orienté dont les nœuds sont des attributs et les arcs définissent les appariements.

De manière formelle, si $\mathcal{A}$ désigne l'ensemble des attributs, le graphe $\mathcal{G}_{\mathtt{a}}$ d'appariements d'attributs (Cf. figure 3.11) est défini par $\mathcal{G}_{\mathtt{a}} = (\mathcal{N}_{\mathtt{a}}, \mathcal{L}_{\mathtt{a}})$, où :

- $\mathcal{N}_{\mathtt{a}} \subseteq \mathcal{A}$ représente l'ensemble des nœuds du graphe d'appariements d'attributs ;
- $\mathcal{L}_{\mathtt{a}} = \left\{(\mathtt{A_s}, \mathtt{a_t}, \alpha_{\mathtt{a}}, \varphi, \mathtt{eL}) \in \mathcal{N}_{\mathtt{a}}^{\mathtt{k}} \times \mathcal{N}_{\mathtt{a}} \times [\mathtt{0}, \mathtt{1}] \times \mathcal{F}_{\mathtt{sem}} \times \mathcal{L}_{\mathtt{e}}\right\}$ représente les noeuds d'appariement d'un ensemble ordonné d'attributs $\mathtt{A_s} \subset \mathcal{N}_{\mathtt{a}}$ avec un attribut cible ($\mathtt{a_t}$) et avec le degré de confiance $\alpha_{\mathtt{a}}$. Un appariement d'attributs est forcément associé à un *eLink* qui représente son *contexte de création*. $\varphi \in \mathcal{F}_{\mathtt{sem}}$ est une expression permettant d'exprimer $\mathtt{a_t}$ en fonction des attributs de $\mathtt{A_s}$. Tous les attributs de $\mathtt{A_s}$ sont associés à la même entité.

En considérant le processus de construction d'ontologies offert par la plate-forme DaFOE, on a par exemple :

- $\mathcal{N}_{\mathtt{a}}$ ={`frequency, rate, relevance, ...`}
- $\mathcal{L}_{\mathtt{a}}$ = {$\mathtt{aL_1}$ = ((`frequency`, `variant_number`), `rate`, 1, "`frequency + variant_number`", $\mathtt{eL_1}$), $\mathtt{aL_2}$ = ((`rate`), `relevance`, 1, "`rate/100`", $\mathtt{eL_2}$), ...}

Le graphe d'appariements d'attributs n'est pas forcément acyclique de manière à permettre à un utilisateur de définir un appariement inverse. Toutefois, du fait de la non-acyclicité de ce graphe, et pour éviter une éventuelle exploration infinie lors de son exploitation, le graphe d'appariement d'attributs est parcouru en largeur avec marquage des nœuds déjà visités.

Dans la suite de ce document, nous utilisons le terme ***aLink*** pour désigner ce type d'appariement.

3.5.4.2 Modèle d'expressions

Le modèle d'expressions permet de caractériser l'ensemble $\mathcal{F}$ des expressions admissibles pour l'appariement d'attributs. A ce modèle d'expression, nous associons un ensemble $\mathcal{F}_{\mathtt{sem}}$ composé des éléments de $\mathcal{F}$ pour lesquels la sémantique de chaque variable est portée par un attribut du modèle. Par exemple, l'expression "`z = x + y`" de $\mathcal{F}$ peut avoir pour interprétation l'expression sémantique "`rate = frequency + variant_number`" de $\mathcal{F}_{\mathtt{sem}}$ dans laquelle les variables `z`, `x` et `y` ont respectivement été

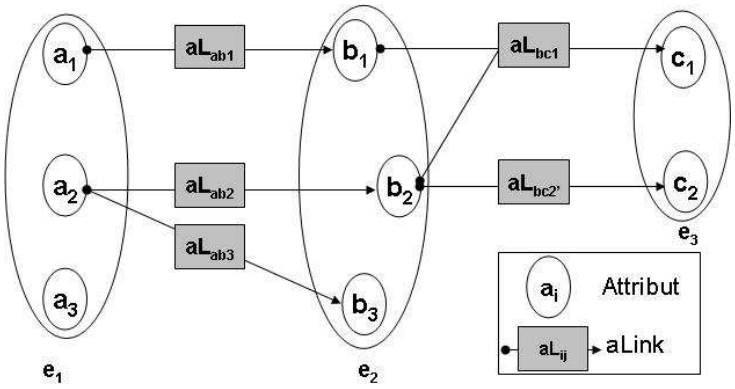

FIGURE 3.11 – Graphe des aLinks.

remplacées par les attributs `rate`, `frequency` et `variant_number`. Comme le montre la figure 3.12, le modèle d'expressions permettant de construire les éléments de $\mathcal{F}_{\texttt{sem}}$ est constitué d'un ensemble d'opérateurs de base pour la manipulation des chaînes de caractères, des nombres réels, des valeurs booléennes, etc.. Toutefois, ces opérateurs de base peuvent être combinés pour obtenir des expressions plus complexes. D'autres propriétés de l'ensemble $\mathcal{F}_{\texttt{sem}}$ seront présentées dans le chapitre 5 dans lequel nous abordons le problème de l'exploitation des mappings pour l'interrogation des données.

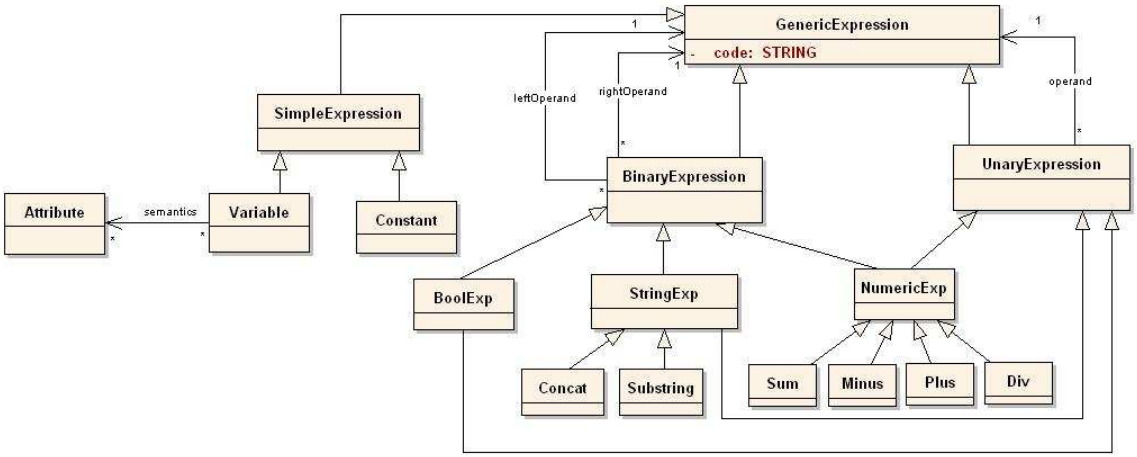

FIGURE 3.12 – Modèle d'expressions.

3.5.4.3 Propriétés d'un aLink

L'appariement d'attributs permet, lors de l'interrogation des entités, d'exploiter les attributs appariés. Les attributs susceptibles d'être utilisés lors d'une telle interrogation correspondent à ceux pour lesquels il est possible de trouver un *aLink* (directement ou par composition de *aLinks*). La composition des *aLinks* n'est possible que pour des *aLinks* consécutifs. Nous parlons alors de *compatibilité des aLinks*.

Comme illustré par la figure 3.11, les appariements entre attributs peuvent être exploités de manière transitive de proche en proche. En effet, considérons les *aLinks* $\texttt{aL}_{\texttt{ab1}} = ((\texttt{a}_1), \texttt{b}_1, \alpha_1, \text{"}\texttt{a}_1\text{"}, \texttt{eL}_1)$,

$\mathtt{aL_{ab2}} = ((\mathtt{a_2}), \mathtt{b_2}, \alpha_2, \text{"}\mathtt{a_2}\text{"}, \mathtt{eL_1})$ et $\mathtt{aL_{bc1}} = ((\mathtt{b_1}, \mathtt{b_2}), \mathtt{c_1}, \alpha_3, \text{"}\mathtt{concat}(\mathtt{b_1}, \mathtt{b_2})\text{"}, \mathtt{eL_2})$ exprimant, d'une part, que l'attribut $\mathtt{b_1}$ (respectivement $\mathtt{b_2}$) est identique à l'attribut $\mathtt{a_1}$ (respectivement $\mathtt{a_2}$), et, d'autre part, que l'attribut $\mathtt{c_1}$ résulte de la concaténation des attributs $\mathtt{b_1}$ et $\mathtt{b_2}$. On souhaite caractériser le *aLink* $\mathtt{aL_{ac}} = (\mathtt{A_s}, \mathtt{a_t}, \alpha, \mathtt{f}, \mathtt{eL})$ exprimant l'appariement composé des attributs $\mathtt{a_1}$ et $\mathtt{a_2}$ avec l'attribut $\mathtt{c_1}$.

Définition 3.5.3

Compatibilité des aLinks

Soient $\mathtt{aL_1} = (\mathtt{A_{s_1}}, \mathtt{a_{t_1}}, \alpha_1, \varphi_1, \mathtt{eL_1})$ *et* $\mathtt{aL_2} = (\mathtt{A_{s_2}}, \mathtt{a_{t_2}}, \alpha_2, \varphi_2, \mathtt{eL_2})$ *deux aLinks associés à des contextes différents (i.e* $\mathtt{eL_1} \neq \mathtt{eL_2}$*).* $\mathtt{aL_1}$ *est compatible avec* $\mathtt{aL_2}$ *si* $\mathtt{a_{t_1}} \in \mathtt{A_{s_2}}$*. En considérant le graphe figure 3.11 par exemple, on peut dire que* $\mathtt{aL_{ab1}}$ *est compatible avec* $\mathtt{aL_{bc1}}$ *mais n'est pas compatible avec* $\mathtt{aL_{bc2'}}$*.*

Compatibilité globale des aLinks

Soient $\mathtt{aL} \in \mathcal{L}_\mathtt{a}$ *et* $\mathtt{v} = (\mathtt{aL_1}, ..., \mathtt{aL_n}) \in \mathcal{L}_\mathtt{a}^\mathtt{n}$ *un n-uplet d'aLinks tous associés à un même contexte (i.e* $\forall \mathtt{i}, \mathtt{j}$ *tels que* $1 \leq \mathtt{i}, \mathtt{j} \leq \mathtt{n}$ *et* $\mathtt{i} \neq \mathtt{j}, \mathtt{aL_i}\,[\mathtt{eL}] = \mathtt{aL_j}\,[\mathtt{eL}]$*).* v *est dit globalement compatible avec* aL *si* $\forall \mathtt{i}, 1 \leq \mathtt{i} \leq \mathtt{n}, \mathtt{aL_i}$ *est compatible avec* aL*. En considérant le graphe de la figure 3.11, on peut dire que* $(\mathtt{aL_{ab1}}, \mathtt{aL_{ab2}})$ *est globalement compatible avec* $\mathtt{aL_{bc1}}$*.*

Un *aLink* pouvant faire intervenir conjointement plus de deux attributs, la composition des *aLinks* ne peut être traitée de la même manière que les *mLinks* ou les *eLinks*. De plus, comme le montre la figure 3.11, les appariements d'attributs peuvent être utilisés de manière transitive de proche en proche. En effet, soit aL un *aLink* globalement compatible avec une famille $(\mathtt{aL_1}, ..., \mathtt{aL_n})$ de *aLinks*, on souhaite caractériser le *aLink* $\mathtt{aL'} = (\mathtt{A'_s}, \mathtt{a'_t}, \alpha', \varphi', \mathtt{eL'})$ exprimant l'appariement composé de la famille $(\mathtt{aL_1}, ..., \mathtt{aL_n})$ de *aLinks* avec le *aLink* aL. Nous introduisons pour cela un opérateur dit de *composition globale* des *aLinks*, noté $\bigcirc_g$ et défini de la manière suivante :

$$\begin{array}{rccl} \bigcirc_\mathtt{g} : & \mathcal{L}_\mathtt{a}^\mathtt{n} \times \mathcal{L}_\mathtt{a} & \longrightarrow & \mathcal{L}_\mathtt{a} \\ & ((\mathtt{aL_1}, ..., \mathtt{aL_n}), \mathtt{aL}) & \longmapsto & \mathtt{aL'} = (\mathtt{A'_s}, \mathtt{a'_t}, \alpha', \varphi', \mathtt{eL'}) \end{array} \quad \text{tel que :}$$

$$\begin{cases} \bullet\ \mathtt{A'_s} = \prod_{\mathtt{i}=1}^{\mathtt{n}} (\mathtt{aL_i}\,[\mathtt{A_s}]) & \text{/* produit cartésien des ensembles d'attributs sources de la } 1^{ere} \text{ opérande */} \\ \bullet\ \mathtt{a'_t} = \mathtt{aL}\,[\mathtt{a_t}] & \text{/* attribut cible de la } 2^{nde} \text{ opérande de la composition */} \\ \bullet\ \alpha' = \min(\bigcup_{\mathtt{i}=1}^{\mathtt{n}} \{\min(\mathtt{aL}\,[\alpha_\mathtt{a}], \mathtt{aL_i}\,[\alpha_\mathtt{a}])\}) & \text{/* minimum des degrés de confiance */} \\ \bullet\ \varphi' = \mathtt{Unfold}(\mathtt{aL}\,[\varphi], (\mathtt{aL_1}\,[\varphi], ..., \mathtt{aL_n}\,[\varphi])) & \text{/* Substitution d'expressions */} \\ \bullet\ \mathtt{eL'} = (\mathtt{aL}\,[\mathtt{eL}]) \circ (\mathtt{aL_1}\,[\mathtt{eL}]) \text{ car } \forall \mathtt{i}, \mathtt{j} \text{ tels que } 1 \leq \mathtt{i}, \mathtt{j} \leq \mathtt{n}, \mathtt{aL_i}\,[\mathtt{eL}] = \mathtt{aL_j}\,[\mathtt{eL}] \end{cases}$$

Par construction de $\bigcirc_\mathtt{g}$, la *composition globale* est une opération fermée. De ce fait, tout au long de ce mémoire, la notion de *aLink* fait à la fois référence aux *aLinks* directs entre une famille de *aLinks* et un *aLink* et les *aLinks* obtenus par composition globale. Notons que cette façon de calculer le degré de confiance du *aLink* résultant de la composition de deux *aLinks* constitue l'approche implémentée par

défaut au sein de la plate-forme DaFOE. Nous verrons d'ailleurs qu'un utilisateur peut modifier ce mode de calcul à condition que la *composition globale* demeure une opération fermée. Par ailleurs, la fonction $\texttt{Unfold} : \mathcal{F}_{\texttt{sem}} \times \mathcal{F}_{\texttt{sem}}^{\texttt{n}} \longrightarrow \mathcal{F}_{\texttt{sem}}$ permet, pour une expression $\varphi' = \texttt{aL}[\varphi]$ et une liste d'expressions $\texttt{exp} = (\texttt{aL}_1[\varphi], ..., \texttt{aL}_\texttt{n}[\varphi])$ par exemple, de substituer récursivement chacune des variables de l'expression φ' en utilisant successivement chacune des expressions de exp équivalente à la variable à substituer. L'opérateur $\bigcirc_\texttt{g}$ ne s'applique qu'aux *aLinks* globalement compatibles.

3.5.5 Appariement transversal

Nous utilisons le terme *appariement transversal* pour désigner les appariements impliquant des éléments de natures différentes. C'est le cas par exemple, lorsqu'on souhaite exprimer le fait qu'une entité d'un modèle source ($\texttt{m}_\texttt{s}$) est appariée avec un attribut dans un modèle cible ($\texttt{m}_\texttt{t}$). La figure 3.13 illustre un appariément transversal où les entités *Homme* et *Femme* sont utilisées pour construire l'attribut sexe de l'entité Personne en disant par exemple que :

- "une *femme* est une *personne* de sexe féminin" ;
- "un *homme* est une *personne* de sexe masculin".

Pour supporter ce type d'appariement, nous ramenons le problème sous la forme d'un appariement d'entités suivi d'un appariement d'attributs.

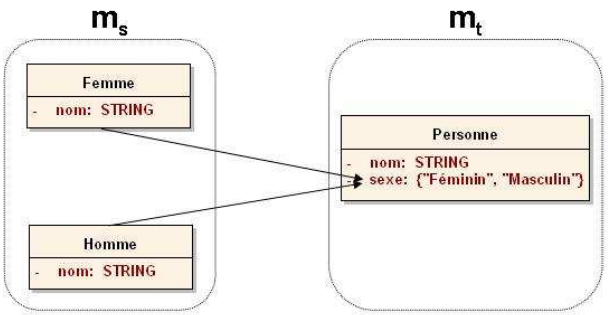

FIGURE 3.13 – Exemple d'appariements transversaux.

Théorème 3.5.5.1

Tout appariement transversal d'une entité vers un attribut peut s'écrire à l'aide d'un eLink et d'un aLink.

Preuve. Par construction, on sait que tout attribut est défini dans le contexte d'une entité et qu'inversement, une entité est complètement définie par la connaissance de son modèle ainsi que de tous ses attributs (en vertu de l'existence de l'application $\texttt{dom} : \mathcal{A} \longrightarrow \mathcal{E}$ définie telle que $\forall \texttt{a} \in \mathcal{A}$, $\exists \texttt{e} \in \mathcal{E}$ vérifiant $\texttt{e} = \texttt{dom}(\texttt{a})$).

Soient $\texttt{m}_1 = (\texttt{E}_1, \texttt{T}_1, \texttt{A}_1, \texttt{I}_1, ...)$ et $\texttt{m}_2 = (\texttt{E}_2, \texttt{A}_2, \texttt{T}_2, \texttt{I}_2, ...)$ deux modèles. Soient $\texttt{e} \in \texttt{E}_1$ et $\texttt{a} \in \texttt{A}_2$. Pour exprimer un appariement de l'entité e avec l'attribut a avec un degré de confiance α, on procède comme suit :

- comme $\mathtt{a} \in \mathtt{A}_2$, on sait qu'il existe $\mathtt{e}' \in \mathtt{E}_2$ tel que $\mathtt{e}' = \mathtt{dom}(\mathtt{a})$ (par construction de l'application dom) ;
- on crée alors un *eLink* de l'entité e vers l'entité $\mathtt{e}'$: $\mathtt{eL} = (\mathtt{e}, \mathtt{e}', \alpha_\mathtt{e}, \mathtt{mL})$ avec $\mathtt{mL} = (\mathtt{its\_model}(\mathtt{e}), \mathtt{its\_model}(\mathtt{e}'), \alpha)$;
- on crée ensuite un *aLink* de l'ensemble vide vers l'attribut a : $\mathtt{aL} = ((), \mathtt{a}, \alpha_\mathtt{a}, \varphi, \mathtt{eL})$;
- $\alpha = \alpha_\mathtt{e} = \alpha_\mathtt{a}$.

■

En appliquant le Théorème 3.5.5.1, la création d'un mapping correspondant à l'exemple de la figure 3.13 s'effectue de la manière suivante :

❶ création d'un *mLink* entre les modèles $\mathtt{m}_1$ et $\mathtt{m}_2$.
$\mathtt{mL}_{12} = (\mathtt{m}_1, \mathtt{m}_2, \alpha_{12})$;

❷ création d'un *eLink* entre les entités Femme et Personne.
$\mathtt{eL}_{\mathtt{FP}} = (\mathtt{Femme}, \mathtt{Personne}, \alpha_{\mathtt{FP}}, \mathtt{mL}_{12})$;

❸ création d'un *eLink* entre les entités Homme et Personne.
$\mathtt{eL}_{\mathtt{HP}} = (\mathtt{Homme}, \mathtt{Personne}, \alpha_{\mathtt{HP}}, \mathtt{mL}_{12})$;

❹ création de deux *aLinks* pour l'attribut sexe de l'entité Personne.
$\mathtt{aLsex}_{\mathtt{FP}} = ((), \mathtt{sexe}, \alpha_{\mathtt{FP}}, \text{"}\mathtt{Feminin}\text{"}, \mathtt{eL}_{\mathtt{FP}})$ et $\mathtt{aLsex}_{\mathtt{HP}} = ((), \mathtt{sexe}, \alpha_{\mathtt{HP}}, \text{"}\mathtt{Masculin}\text{"}, \mathtt{eL}_{\mathtt{HP}})$;

❺ création de deux *aLinks* pour l'attribut nom de l'entité Personne.
$\mathtt{aLnom}_{\mathtt{FP}} = ((\mathtt{nom}), \mathtt{nom}, \alpha_{\mathtt{FP}}, \text{"}\mathtt{nom}\text{"}, \mathtt{eL}_{\mathtt{FP}})$ et $\mathtt{aLnom}_{\mathtt{HP}} = ((\mathtt{nom}), \mathtt{nom}, \alpha_{\mathtt{HP}}, \text{"}\mathtt{nom}\text{"}, \mathtt{eL}_{\mathtt{HP}})$;

Corollaire 3.5.5.1

$\forall \mathtt{aL} \in \mathcal{L}_\mathtt{a}$, si $\mathtt{aL}[\mathtt{A_s}] = ()$ alors $\mathtt{aL}[\varphi]$ est une expression constante du modèle d'expression présenté dans la section 3.5.4.2.

3.6 Évolution du méta-modèle noyau

Dans les sections précédentes, nous avons présenté une formalisation de la représentation des modèles et des mappings. Nous avons regroupé ces deux formalisations au sein d'un modèle unique appelé *méta-modèle noyau* (*Core Metamodel*) et illustré par la figure 3.14 sur laquelle on trouve à la fois les concepts liés à la représentation des modèles (*Model, Entity, Attribute, etc.*) et les concepts liés à la représentation des mappings (*mLink, eLink, aLink, GenericExpression*).

A la différence des autres travaux portant sur la représentation des mappings [Renée Miller, 2000, Ling et al., 2001, Horrocks et al., 2004] qui supposent non seulement l'existence d'un ensemble fini de constructeurs de mappings, mais également que ces constructeurs sont figés, nous proposons une approche permettant à la fois de créer de nouveaux constructeurs de mappings, mais aussi d'enrichir dynamiquement les constructeurs existants. Dans cette section, nous illustrons le besoin d'extension de la représentation des mappings et proposons une approche de prise en compte de la composition lors de l'extension de cette représentation des mappings.

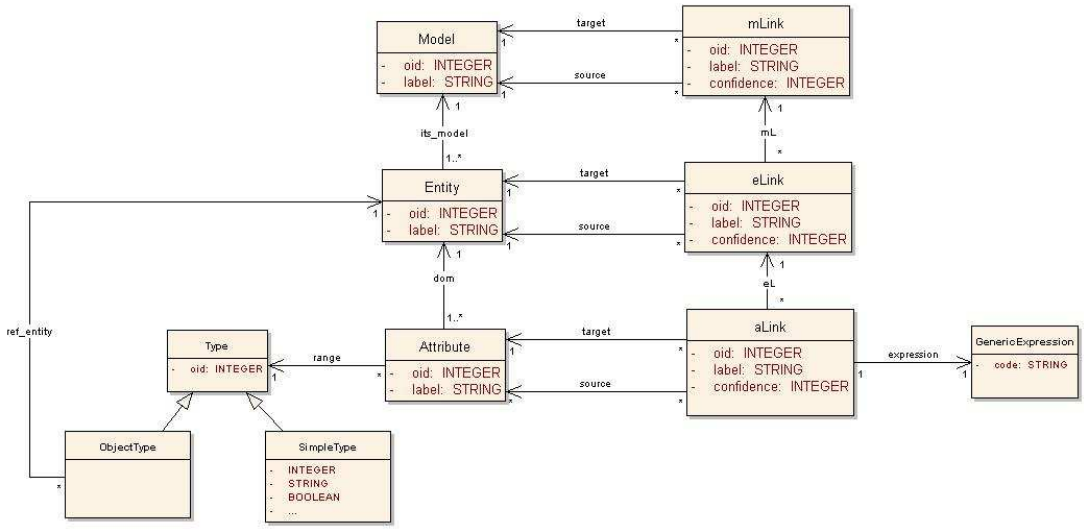

FIGURE 3.14 – Méta-modèle noyau.

3.6.1 Illustration

Les constructeurs de mappings que nous avons présentés dans la section 3.5 utilisent une valeur numérique pour représenter le degré de confiance d'un mapping. Il s'agit de l'approche classique de gestion des mappings flous. Cependant, que se passe-t-il si un utilisateur souhaite créer une propriété floue permettant d'annoter chaque mapping par une *valeur de qualité* (*"faible", "moyen", "bien", "excellent", etc.* par exemple)? Dans notre approche, nous proposons de représenter le modèle de mapping lui-même au sein d'un modèle : le *méta-modèle de mappings*. De cette manière, les constructeurs de mappings tels que *mLink, eLink et aLink* constituant le modèle de mappings, représentent le sous-ensemble de base des instances du méta-modèle de mappings. Grâce à cette possibilité de représenter les constructeurs de mappings comme des instances, le modèle de mappings devient donc dynamiquement extensible et son extension se fait sans modification manuelle des programmes sous-jacents au modèle de mappings.

Par ailleurs, le modèle de mappings étant aussi lié à la structure du méta-modèle (Entité-Attribut) permettant de décrire les modèles, et puisque dans notre approche nous ne faisons pas l'hypothèse d'un méta-modèle figé, nous avons donc besoin d'un niveau plus abstrait constitué d'un méta-méta-modèle pour décrire le méta-modèle lui-même. Dans nos travaux, nous avons utilisé un méta-méta-modèle générique de manière à pouvoir à la fois représenter les modèles et les constructeurs de mappings. En d'autres termes, le méta-modèle de mappings est inclu dans le méta-méta-modèle. L'architecture illustrant l'évolution du *méta-modèle noyau* peut être résumée par le schéma de la figure 3.15. La figure 3.16, pour sa part, présente le méta-méta-modèle que nous avons utilisé pour décrire le *méta-modèle noyau*. La classe *MetaEntity* permet de représenter les concepts du *méta-modèle noyau* tels que *Entity, Attribute, mLink, eLink, etc.* tandis que la classe *MetaAttribute* est utilisée pour créer les attributs (*oid, label, etc.*) de ces concepts.

La possibilité d'extension du modèle de mappings n'est pas sans conséquence sur l'infrastructure globale. En effet, du fait de la transitivité des mappings, les extensions apportées au modèle de mappings doivent être prises en compte lors de la composition des mappings.

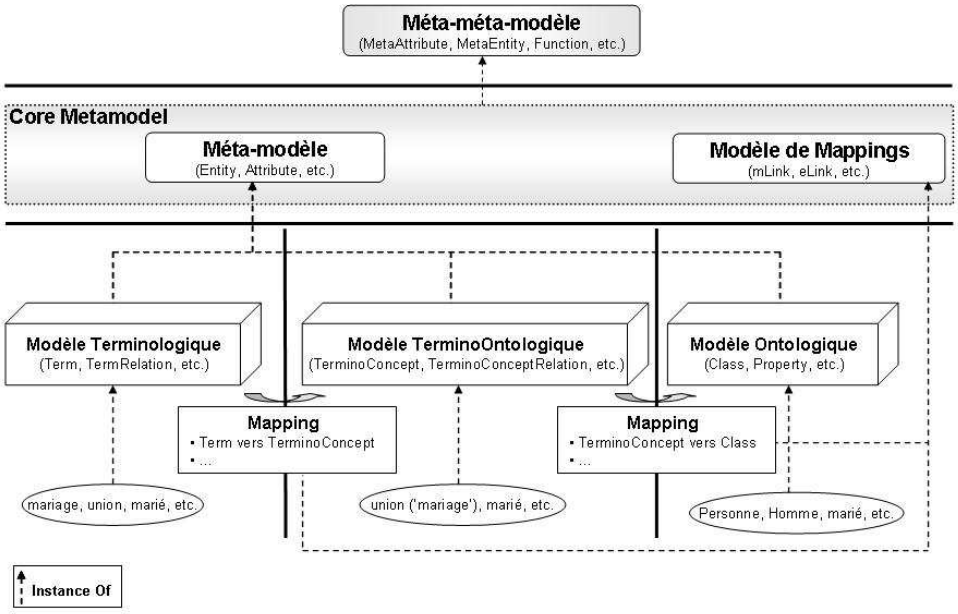

FIGURE 3.15 – Architecture d'évolution du modèle de mappings.

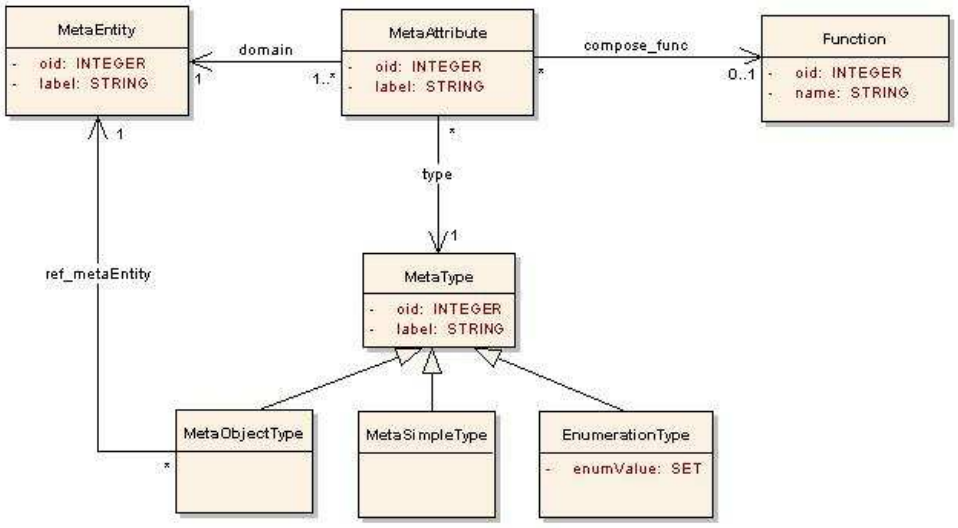

FIGURE 3.16 – Méta-méta-modèle.

3.6.2 Prise en compte de la composition

Comme nous l'avons vu dans la section 3.5, la composition est applicable aux différents types de constructeurs de mappings. Nous avons d'ailleurs montré, pour chacun de ces constructeurs, une manière de synthétiser leur composition respective. Cette modélisation statique de la composition n'est pas applicable pour la gestion de la composition d'attributs dynamiquement ajoutés (la *qualité* d'un mapping par exemple). En effet, comme illustré dans la figure 3.17, on souhaite calculer, à partir de deux valeurs q_{12} et q_{23} décrivant les qualités respectives de deux *mLinks*, la valeur q_{13} de la qualité résultant de q_{12} et q_{23}. Pour y parvenir, nous généralisons la représentation de la composition et proposons de la modéliser

au sein du méta-méta-modèle comme le montre la figure 3.16. Sur cette figure, la classe `Function` fournit un ensemble de fonctions de base (similaires au modèle d'expressions que nous avons présenté dans la section 3.5). Parmi ces fonctions de base, nous pouvons par exemple citer l'addition, la soustraction, la division, le maximum, le minimum, etc..

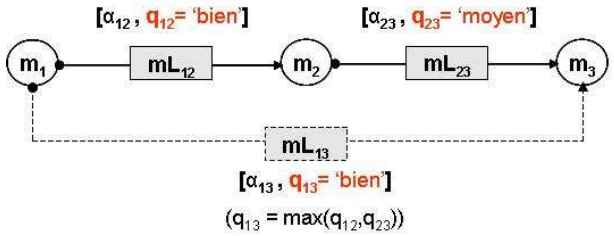

FIGURE 3.17 – Composition de mLinks avec évolution du modèle de mappings.

Pour supporter l'illustration présentée par la figure 3.17, notre approche est mise en œuvre de la manière suivante :

1. on crée, pour la classe `EnumerationType` de la figure 3.18, une instance `qualityType` représentant un l'ensemble ordonnée constitué des valeurs *"faible", "moyen", "bien", "excellent", etc.* ;
2. on créé, pour la classe `MetaAttribute` de la figure 3.18, le *méta-attribut* `"quality"` de type *qualityType*. Ce *méta-attribut* référence la fonction `min` de la classe `Function` de la figure 3.18. Ce qui signifie que, lors de la composition de deux *mLinks* pour lesquels on connaît les valeurs prises par le *méta-attribut* `quality`, la valeur résultant de la composition de ces deux valeurs correspond au maximum de ces deux valeurs. La relation d'ordre utilisée est celle d'apparition des éléments constituant le type énuméré.

De façon analogue, il est possible de modifier la manière de calculer la composition de différents types de constructeurs de mappings. En effet, dans la section 3.5, nous avons proposé une méthode par défaut de calcul du degré de confiance des appariements. Par défaut, ce calcul du degré de confiance d'une composition de degrés de confiance utilisait la fonction minimun. Pour modifier cette méthode de calcul, l'utilisateur peut utiliser l'une des fonctions instanciées dans la classe *Function* de la figure 3.18.

3.7 Conclusion

Dans ce chapitre, nous avons présenté notre approche de représentation des mappings. En nous positionnant en faveur des approches par méta-modèle, nous avons distingué trois principaux types d'appariements suivant la nature des éléments (modèle, entité, attribut) à apparier. Toutefois, notre approche ne se limite pas à ces principaux types d'appariements. En effet, elle permet non seulement de créer de nouveaux constructeurs d'appariements mais également de modifier les constructeurs existants grâce à un méta-méta-modèle. Constatant que la composition est une opération fondamentale pour les mappings, nous avons caractérisé la composition pour les types d'appariements proposés. Toutefois, cette caractérisation ne représente qu'un calcul par défaut, un utilisateur pouvant faire évoluer le modèle de

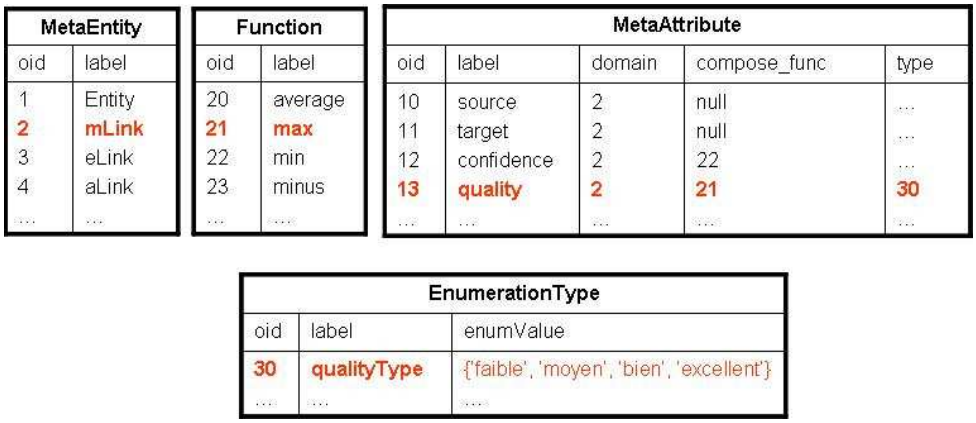

FIGURE 3.18 – Composition des mLinks .

mappings. Afin d'adapter cette caractérisation à une éventuelle évolution du modèle de mappings, nous avons représenté les fonctions de composition au sein d'un méta-méta-modèle. Par ailleurs, compte tenu de la cardinalité des appariements d'attributs, nous avons introduit un opérateur, dit de *composition globale*. Ces opérateurs de composition sont particulièrement utilisés pour une interrogation des données en exploitant les mappings. Mais avant d'aborder la question de l'exploitation de ces mappings, nous présentons, dans le chapitre suivant, l'approche que nous avons suivie pour intégrer les propositions de ce chapitre dans un environnement persistant de bases de données.

Chapitre 4

Transformation en environnement persistant

Sommaire

Résumé. Dans le chapitre 2, nous avons vu que les BDBM permettent d'expliciter la sémantique des données contenues dans la base de données. Dans ce chapitre, nous présentons l'extension que nous proposons aux BDBM pour la prise en compte des mappings. L'objectif est de proposer une infrastructure capable de stocker les mappings conformément à l'approche de modélisation que nous avons présentée dans le chapitre précédent.

4.1 Introduction

Dans le chapitre précédent, nous avons présenté notre approche de représentation des modèles et des mappings entre modèles, en évoquant en particulier la volonté de disposer d'un mécanisme permettant la création de nouveaux constructeurs de mappings. Dans ce chapitre, nous nous intéressons à la transposition de ces résultats dans un contexte persistant de base de données. Pour cela, nous discutons, dans la section 4.2, de la capacité des architectures de bases de données actuelles à supporter la gestion des mappings. Au vu des insuffisances des BDBM existantes, nous proposons (Cf. section 4.3) de nous appuyer sur l'architecture de la BDBM OntoDB [Dehainsala et al., 2007a] pour laquelle nous proposons une extension pour la représentation des mappings [Téguiak et al., 2012b]. Pour permettre cette prise en compte des mappings dans une architecture similaire à celle d'OntoDB, nous avons manipulé ses parties *méta-schéma* et *ontologie* et validé cette extension par une application de nos propositions au sein de la plate-forme DaFOE (Cf. section 4.5).

4.2 Choix d'une architecture de base de données

Afin de valider notre approche, nous avons besoin, d'une part, d'une infrastructure permettant d'encoder les modèles associés aux différentes étapes du processus de construction d'ontologies, et, d'autre part, d'un langage permettant de manipuler les éléments qui composent cette infrastructure. En utilisant les graphes d'appariements présentés dans le chapitre précédent (Cf. section 3.5), nous montrons, dans cette section, que les BDBM de type 3, grâce à leur capacité à étendre à la fois les modèles et les méta-modèles sont mieux adaptés à notre besoin d'évolution des modèles au sein de la plate-forme DaFOE. De plus, comme nous l'avons présenté dans le chapitre précédent, notre approche de représentation des mappings permet de faire évoluer le modèle de mappings en offrant notamment la possibilité de créer de nouveaux constructeurs de mappings ou de modifier les constructeurs existants. Les constructeurs de mappings étant modélisés dans le *méta-modèle noyau*, nous avons donc besoin d'une architecture de base de données dans laquelle le méta-modèle n'est pas figé.

4.2.1 BDBM de type 1

Dans le chapitre chapitre 2 (Cf. section 2.7.1.2), nous avons vu que les BDBM de type 1 utilisent un unique schéma (constitué d'une unique table) pour la représentation des modèles et des instances. Ainsi, pour implémenter les graphes d'appariements (que nous avons présentés dans la section 3.5 du chapitre précédent) dans une BDBM de type 1 comme RDF, par exemple, nous avons identifié quelques règles à appliquer :

- utiliser RDF schéma comme méta-modèle pour la représentation des modèles. Ainsi, le triplet `(Term, Type, Entity)` signifierait que le concept `Term` est une classe RDF (que nous appelons Entité conformément au *méta-modèle noyau* que nous avons proposé) ;
- étendre RDF Schéma en y rajoutant des concepts représentant chaque constructeur de mappings (*mLink, eLink, aLink*) ;
- étendre RDF schéma en y ajoutant des propriétés permettant de représenter les éléments des ensembles $\mathcal{L}_{\mathtt{m}}$, $\mathcal{L}_{\mathtt{e}}$ et $\mathcal{L}_{\mathtt{a}}$ représentant les arcs des graphes d'appariements. Ainsi, "`rdfs : source`"'

permettrait, par exemple, d'identifier le modèle source d'un *mLink* tandis que "`rdfs : eL`" permettrait d'identifier le *mLink* associé à un *eLink* créé.

Dans la figure 4.1, nous avons utilisé les différents modèles de la plate-forme DaFOE pour illustrer l'application de ces règles . Cependant, comme nous l'avons précisé dans le chapitre 2 (Cf. section 2.7.1.2), cette approche de stockage des instances et des modèles passe moins à l'échelle que les autres types de BDBM du fait de l'utilisation d'une unique table de stockage et sur laquelle il est nécessaire de faire de nombreuses opérations d'auto-jointure [Alexaki et al., 2001].

Subject	Predicat	Object
TerminoOntology	rdf:type	rdfs:Model
Ontology	rdf:type	rdfs:Model
TerminoConcept	rdf:type	rdfs:Entity
TerminoConcept	rdfs:Model	rdfs:TerminoOntology
Class	rdf:type	rdfs:Entity
Class	rdfs:Model	rdfs:Ontology
rate	rdf:type	rdfs:Attribute
rate	rdfs:domain	TerminoConcept
rate	rdfs:range	xsd:integer
relevance	rdf:type	rdfs:Attribute
relevance	rdfs:domain	Class
relevance	rdfs:range	xsd:integer
mL_1	rdf:type	rdfs : mLink
mL_1	rdfs:source	TerminoOntolgy
mL_1	rdfs:target	Ontology
eL_1	rdf:type	rdfs:eLink
eL_1	rdfs:mL	mL_1
eL_1	rdfs:source	TerminoConcept
eL_1	rdfs:target	Class
aL_1	rdf:type	rdfs:aLink
aL_1	rdfs:eL	eL_1
aL_1	rdfs:source	rate
aL_1	rdfs:target	relevance
aL_1	rdfs:expression	rate/100
...	...	...

FIGURE 4.1 – BDBM de type 1.

4.2.2 BDBM de type 2

Dans le chapitre 2 (Cf. section 2.7.1.2), nous avons vu que les BDBM de type 2 stockent séparément les données de niveau modèle (ou schéma) et les données de niveau instance. Les modèles sont stockés conformément à un méta-modèle représenté par un ensemble de tables prédéfinies dans la BDBM. Comme toutes les autres tables, elles sont stockées dans la méta-base du SGBD. Ces tables permettent

de représenter des modèles constitués de classes et de propriétés. Par contre, aucune solution n'est fournie pour ajouter de nouvelles tables pour la définition de modèles particuliers. Ainsi, pour représenter le modèle de mappings présenté dans le chapitre précédent, il est nécessaire de modifier la BDBM en ajoutant manuellement les tables nécessaires (Cf. figure 4.2). Cependant, puisque les tables sont ajoutées manuellement, les langages et outils associés à la BDBM ne peuvent plus être utilisés puisqu'ils ne permettent pas l'exploitation de ces nouvelles tables sauf en écrivant des programmes d'accès spécifiques qui nécessitent la connaissance de la structure interne de la BDBM. Comme nous allons le voir dans la section suivante, les BDBM de type 3 résolvent ce problème.

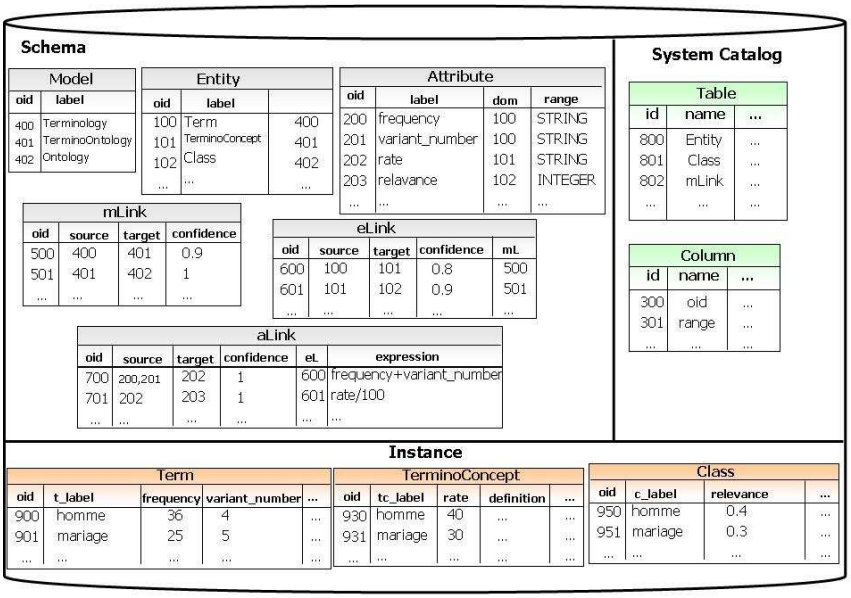

FIGURE 4.2 – BDBM de type 2.

4.2.3 BDBM de type 3

Grâce à la disponibilité d'un méta-méta-modèle permettant de représenter le méta-modèle, les BDBM de type 3 peuvent servir de base pour une gestion des mappings, conformément à notre approche. A notre connaissance, aucune BDBM de type 3 ne propose une solution à la problématique de l'évolution de la représentation des mappings. Nous proposons d'équiper ce type de BDBM de mécanismes pour la prise en compte de cette problématique.

Pour proposer notre infrastructure, nous nous sommes appuyés sur la BDBM OntoDB. Les raisons fondamentales de ce choix sont les suivantes :

1. tout d'abord, et comme précédemment observé, les BDBM de type 1 et 2 ne supportent qu'un unique modèle d'ontologie (RDFS, OWL, PLIB,...). Or, dans notre étude de cas sur la modélisation du processus de construction d'ontologies, chaque étape de modélisation dispose de son propre modèle. Nous avons donc besoin d'une BDBM multi-modèles. La représentation de plusieurs modèles dans une BDBM a été étudiée dans les travaux visant à stocker conjointement les modèles PLIB et OWL dans la BDBM OntoDB [Fankam, 2009] ;

2. ensuite, chacune des étapes du processus de construction d'ontologies est modélisée par les experts de chaque domaine ; les linguistes proposant une modélisation de la terminologie, et les ontologues, un modèle d'ontologies. De ce fait, ces modèles sont susceptibles d'évoluer. Nous souhaitons pouvoir supporter ces évolutions au niveau des mappings. En effet, comme nous l'avons évoqué précédemment, l'évolution d'un modèle est susceptible d'entraîner la création de nouveaux types de constructeurs de mappings. La partie *méta-schéma* d'OntoDB qui permet l'évolution du modèle d'ontologie est donc adaptée à notre besoin ;
3. enfin, face à une forte volumétrie comme c'est susceptible d'être le cas dans la construction d'ontologies à partir de textes, l'architecture de BDBM choisie doit pouvoir supporter le passage à l'échelle. Dans ce cas, elle doit nécessairement être soit de type 2, soit de type 3. Les tests de performance menés sur OntoDB [Dehainsala et al., 2007a] ont montré que cette BDBM répond suffisamment à ce besoin.

4.3 Adaptation des BDBM de type 3 pour la gestion des mappings

Nous avons vu que les BDBM de type 3, grâce à leur partie méta-schéma, répondent au besoin d'évolution du méta-modèle. Il est donc envisageable d'étendre la BDBM OntoDB pour supporter les mappings. Cependant, la BDBM OntoDB, conçue pour stocker les ontologies et leurs instances dans le contexte de l'ingénierie, s'appuie sur un méta-modèle et un méta-schéma complexes comparés aux besoins de la plate-forme DaFOE. Ainsi, plutôt que de prendre OntoDB dans sa globalité et de l'étendre de manière à supporter les mappings, nous avons, dans le cadre du projet DaFOE4App, proposé de concevoir une BDBM similaire à OntoDB mais dont le méta-modèle et le méta-schéma sont simplifiés conformément aux besoins de la plate-forme DaFOE. Cette nouvelle BDBM est ensuite enrichie en y ajoutant les aspects propres à la gestion des mappings. Dans cette section, nous présentons notre adaptation des BDBM de type 3 pour la prise en compte des mappings. En comparaison à la BDBM OntoDB présentée dans le chapitre 2 (Cf. section 2.7.2), nous avons, d'une part, étendu la partie schéma de la BDBM en y ajoutant la représentation des mappings. D'autre part, nous avons étendu la partie méta-schéma pour la prise en compte des fonctions permettant notamment d'exprimer la composition de méta-attributs, nécessaire au calcul de la composition des mappings telle que nous l'avons définie dans le chapitre précédent. Ces constructeurs de mappings sont créés comme instances du méta-schéma de manière à favoriser aussi bien la création dynamique de nouveaux constructeurs de mappings que la modification des constructeurs existants.

4.3.1 Construction de l'infrastructure

La mise en œuvre de notre approche consiste à créer l'infrastructure générique de stockage des constructeurs de mappings. En d'autres termes, il s'agit de décrire la démarche de création des différentes parties de la BDBM. Cette infrastructure de BDBM est obtenue en trois principales étapes.

1. **Mise en place de la partie méta-schéma.** Nous commençons par créer la structure de la partie méta-schéma car elle est statique. Ensuite, nous utilisons, par exemple, les instructions SQL du tableau 4.1 pour créer de nouvelles instances dans les tables *MetaEntity* et *MetaAttribute* de la

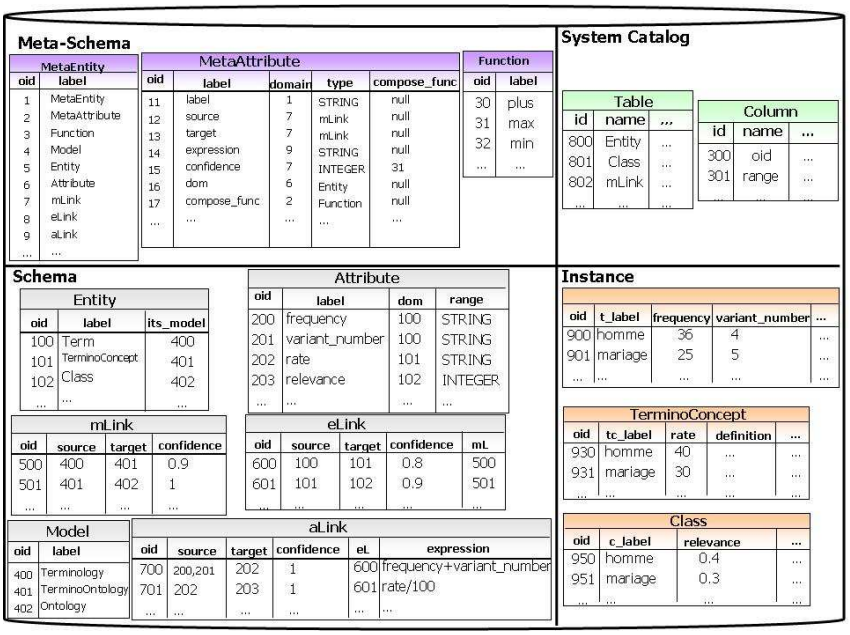

FIGURE 4.3 – Support des mappings dans la BDBM.

figure 4.3. Ces instructions créent les concepts du méta-modèle noyau. Par conséquent, la structure physique de persistance pour chaque type de constructeurs de mappings est automatiquement créée dans la partie schéma de la BDBM. La règle utilisée est de créer, pour chaque méta-entité, une table dont les colonnes correspondent aux méta-attributs de la méta-entité. Notons que, pour faciliter la lecture, les instructions du tableau utilisent les noms des objets comme clés étrangères au lieu de leurs identifiants.

Tableau 4.1 – Création du *Core Metamodel*

```
INSERT INTO MetaEntity (label) VALUES("mLink")
INSERT INTO MetaEntity (label) VALUES("eLink")
INSERT INTO MetaEntity (label) VALUES("aLink")
INSERT INTO MetaEntity (label) VALUES("Model")
...
INSERT INTO MetaAttribute (label, domain, type) VALUES(source,mLink,Model)
...
INSERT INTO MetaAttribute (label, domain, type) VALUES(target,mLink,Model)
...
INSERT INTO MetaAttribute (label, domain, type) VALUES(confidence,mLink,INTEGER)
...
```

2. **Création des modèles.** La création des modèles consiste à instancier les tables *Entity*, *Attribute* et *Model* de la partie schéma (Cf. figure 4.3). Pour chaque entité créée (`Term, TerminoConcept, Class`, etc.) dans la partie schéma, la structure physique (les tables `Term, TerminoConcept, Class`, etc.) de persistance des données est créée dans la partie instance de la BDBM. La règle utilisée est de créer, pour chaque entité, une table dont les colonnes correspondent aux attributs de

l'entité.

3. **Création des mappings.** La création des mappings consiste à instancier les tables *mLink, eLink, aLink* représentant les différents constructeurs de mappings (Cf. figure 4.3). Cette représentation explicite des mappings favorise leur traçabilité.

4.3.2 Prise en compte de la composition

Comme nous l'avons évoqué dans le chapitre précédent (Cf. section 3.6.2), la composition des mappings est une préoccupation sous-jacente à la gestion dynamique des mappings. Nous avons notamment proposé d'utiliser une base de fonctions exploitable pour calculer la composition des mappings. Dans le cas d'une BDBM, cette base de fonctions, essentiellement constituée de fonctions élémentaires du langage SQL (*addition, soustraction, minimum, maximum, etc.*) est stockée dans la table `Function` de la partie *méta-schéma* (Cf. figure 4.3).

En guise d'illustration, le tableau 4.2 présente la création du degré de confiance des constructeurs de mapping pour lesquels la fonction *maximum* est associée pour le calcul de la composition des degrés de confiance. Rappelons cependant que cette approche ne s'applique qu'aux méta-attributs de type simple ; ceux représentant les clés étrangères étant directement gérés par un programme, car la valeur prise par la clé étrangère est la même que celle de l'objet référencé.

Tableau 4.2 – Illustration de la persistance des compositions de mappings

```
INSERT INTO MetaAttribute (label, domain, type, compose_func)
VALUES(confidence,mLink,INTEGER, "max")
...
INSERT INTO MetaAttribute (label, domain, type, compose_func)
VALUES(confidence,mLink,INTEGER, "max")
...
```

4.4 Contrôle de cohérence

Dans le chapitre précédent, nous avons vu que la plate-forme DaFOE offre la possibilité d'étendre le processus de construction d'ontologies en créant notamment de nouvelles étapes auxquelles sont associées des modèles, des entités, des attributs et des mappings. Cette évolution des modèles et des mappings n'est pas sans conséquence sur l'infrastructure de persistance que représente la BDBM. Dans, cette section, nous proposons un ensemble de règles de gestion permettant, lors d'une évolution des modèles et/ou des mappings, de préserver la cohérence de la BDBM.

4.4.1 Évolution des modèles

Dans nos travaux, l'évolution des modèles porte sur la modification de la structure des modèles. Deux opérations principales sont utilisées à cet effet. Nous avons, d'une part, l'opération d'ajout d'un

élément (modèle, entité ou d'un attribut), et, d'autre part, nous avons la suppression d'un élément. Dans cette section, nous présentons l'impact de ces opérations sur les mappings.

1. *Opération d'ajout.*
   L'aspect d'appariement automatique ne rentrant pas dans le cadre de notre travail, pour chaque élément (modèle, entité ou attribut) créé par un utilisateur, cet utilisateur se doit de créer (par instanciation) les appariements nécessaires à l'insertion de l'élément créé dans le graphe de mappings. Ces appariements qui décrivent des équivalences conceptuelles, sont ensuite exploités, par exemple, lors de l'interrogation des modèles.
2. *Opération de suppression.*
   Lors de la suppression d'un élément (modèle, entité ou d'un attribut), tous les appariements impliquant l'élément supprimé sont également supprimés. En effet, dans le graphe de mappings, les appariements représentent les arcs du graphe et par conséquent, la suppression d'un nœud du graphe entraîne la suppression des arcs auxquels le nœud supprimé était rattaché.

4.4.2 Évolution de mappings

Dans la plate-forme DaFOE, l'évolution des mappings se manifeste, soit par la création ou la suppression d'appariements, soit dans le cas d'un *aLink*, par exemple, par la modification de l'expression décrivant l'attribut cible en fonction des attributs sources.

4.4.2.1 Création d'un appariement

Comme pré-condition à la création d'un appariement, une clause dite de non contradiction des appariements doit être vérifiée. En effet, en considérant les modèles terminologique (TM) et termino-ontologique (TOM) de la plate-forme DaFOE, le tableau 4.3 illustre quelques appariements contradictoires.

Tableau 4.3 – Exemple d'appariements contradictoires.

	TM → TOM
mLink	$\texttt{mL}_1$ = (TM, TOM, 0.9) $\texttt{mL}_2$ = (TM, TOM, 1)
eLink	$\texttt{eL}_1$ = (Term, TerminoConcept, 0.8, $\texttt{mL}_1$) $\texttt{eL}_2$ = (Term, TerminoConcept, 1, $\texttt{mL}_1$)
aLink	$\texttt{aL}_1$ = ((frequency, variant_number), rate, 1,′ frequency + variant_number′, $\texttt{eL}_1$) $\texttt{aL}_2$ = ((frequency), rate, 1,′ frequency′, $\texttt{eL}_1$)

1. Les appariements $\texttt{mL}_1$ et $\texttt{mL}_2$ sont contradictoires car ils impliquent les mêmes modèles mais avec des degrés de confiances différents. Pour interdire ce type d'appariement, nous vérifions, avant de créer un *mLink*, l'unicité du couple constitué par les modèles source et cible.

2. Les *eLinks* $\texttt{eL}_1$ et $\texttt{eL}_2$ sont contradictoires car ils proposent, dans le contexte d'un même *mLink* $\texttt{mL}_1$, deux manières différentes d'apparier l'entité Term du modèle terminologique à l'entité TerminoConcept du modèle termino-ontologique. En imposant, dans notre spécification de la BDBM

de la plate-forme DaFOE, une contrainte d'unicité des triplets formés par les entités source et cible du *eLink* ainsi que le *mLink* identifiant le contexte de création du *eLink*, nous interdisons la création des *eLinks* contradictoires.

3. De la même manière, les *aLinks* $\texttt{aL}_1$ et $\texttt{aL}_2$ sont contradictoires car ils proposent, dans le contexte d'un même *eLink* $\texttt{eL}_1$, deux manières différents de calculer la valeur prise par un tuple pour l'attribut cible `rate`. En imposant, dans notre spécification de la BDBM de la plate-forme DaFOE, une contrainte d'unicité des couples formés par l'attribut cible du *aLink* et le *eLink* identifiant le contexte de création du *aLink*, nous interdisons la création des *aLinks* contradictoires.

La création d'un appariement n'est pas sans conséquence sur la BDBM. Elle impose, par exemple, d'adapter tout ou partie des tuples contenus dans la table correspondant à l'entité cible en fonction de l'appariement créé. Par exemple, après la création d'un *eLink* (ainsi que d'un ou plusieurs *aLink(s)* associé(s)), il est nécessaire, pour préserver la cohérence de la BDBM, de peupler la table correspondant à l'entité cible du *eLink* créé en utilisant les tuples contenus dans la table correspondant à l'entité source de ce *eLink*. La création d'un *aLink*, pour sa part, ne nécessite que la mise à jour, pour chaque tuple, de la valeur contenue dans la colonne correspondant à l'attribut cible du *aLink*.

4.4.2.2 Suppression d'un appariement

La suppression d'un appariement pose également le problème de cohérence de données. En effet, la suppression d'un *eLink*, par exemple, entraîne la suppression, dans la table correspondant à l'entité cible du *eLink*, de tous les tuples ayant été construit en utilisant l'appariement supprimé. Pour préserver la cohérence des données suite à la suppression d'un *aLink*, pour tous les tuples de la table correspondant à l'entité cible du *eLink* associé au *aLink* à supprimer, la valeur prise pour la colonne correspondant à l'attribut cible du *aLink* supprimé est remplacée par `NULL`.

4.4.2.3 Modification d'un appariement

Pour les besoins de la plate-forme DaFOE, la modification d'appariements consiste en la modification, pour un *aLink* donné, de l'expression décrivant l'attribut cible en fonction des attributs sources. Cette modification pose également le problème de cohérence des données suite à la modification de l'expression. Pour préserver la cohérence des données, nous recalculons, pour chaque tuple créé en utilisant cet *aLink*, la nouvelle valeur prise pour la colonne correspondant à l'attribut cible du *aLink*.

Tous ces différents scénarios d'évolution de mappings ont été formalisé par des règles et contraintes ECA (Event Condition Action) [Berndtsson and Lings, 1995].

4.5 Application à DaFOE

Cette section illustre l'application de nos propositions dans le cas d'une construction d'ontologies à partir de textes. La figure 4.4 résume le résultat des étapes de construction décrites ci-dessous.

4.5.1 Démarche

Nous avons vu que la construction d'ontologies à partir de textes dans la plate-forme DaFOE passe par trois principales étapes :

- une étape d'analyse terminologique, dédiée à l'analyse linguistique des termes et relations extraits du corpus de textes ;
- une étape d'analyse termino-ontologique, nécessaire à la désambiguïsation des termes et des relations entre les termes ;
- une étape d'analyse ontologique, permettant de construire une ontologie représentée en terme de classes et de propriétés.

A chacune de ces étapes nous avons associé un modèle permettant de stocker les données manipulées dans ces étapes.

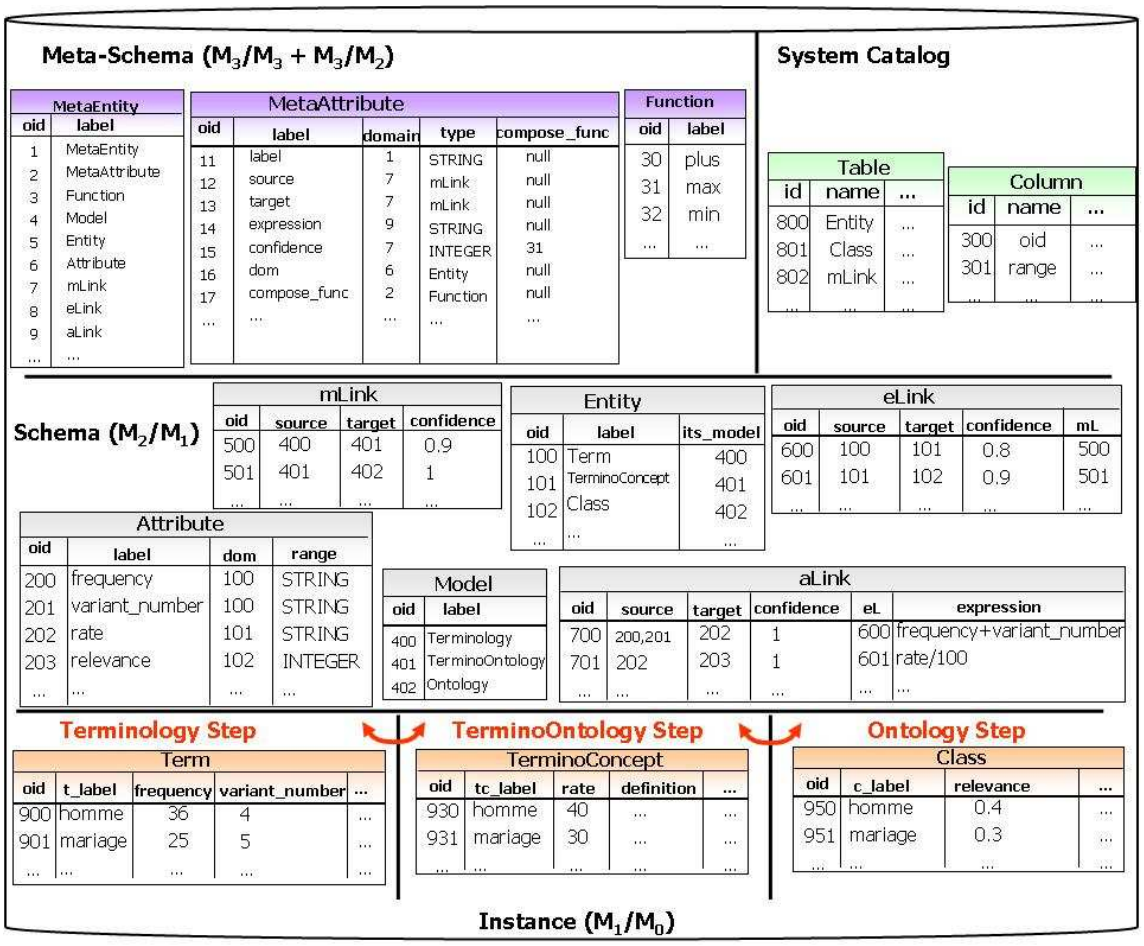

FIGURE 4.4 – Gestion des mappings dans le projet DaFOEApp.

4.5.2 Mise en œuvre

L'application de notre approche permet de construire la structure de persistance présentée sur la figure 4.4. L'implémentation de cette structure de persistance consiste à apparier les modèles des différentes étapes en utilisant des constructeurs de mappings. Dans le cas de la construction d'ontologies à partir de textes telle que proposée dans la plate-forme DaFOE, deux principaux mappings ont été créés : un premier mapping permet de produire des instances du modèle termino-ontologique (TOM) à partir des instances du modèle terminologique (TM) tandis qu'un second mapping permet de produire des instances du modèle ontologique (OM) à partir de ceux du modèle termino-ontologique (TOM). Ces mappings sont illustrés dans le tableau 4.4.

Tableau 4.4 – Exemple de mappings.

	TM → TOM	**TOM → OM**
mLink	$\texttt{mL}_1 = (\texttt{TM}, \texttt{TOM}, \texttt{0.9})$	$\texttt{mL}_2 = (\texttt{TOM}, \texttt{OM}, 1)$
eLink	$\texttt{eL}_1 = (\texttt{Term}, \texttt{TerminoConcept}, \texttt{0.8}, \texttt{mL}_1)$ $\texttt{eL}_2 = (\texttt{TermRelation}, \texttt{TerminoConcept–Relation}, \texttt{0.9}, \texttt{mL}_1)$ ...	$\texttt{eL}_{11} = (\texttt{TerminoConcept}, \texttt{Class}, \texttt{0.9}, \texttt{mL}_2)$ $\texttt{eL}_{12} = (\texttt{TerminoConceptRelation}, \texttt{Property}, \texttt{0.8}, \texttt{mL}_2)$ ...
aLink	$\texttt{aL}_{21} = ((\texttt{t\_label}), \texttt{tc\_label}, 1, \texttt{'t\_label'}, \texttt{eL}_1)$ $\texttt{aL}_{22} = ((\texttt{frequency}, \texttt{variant\_number}), \texttt{rate}, 1, \texttt{'frequency + variant\_number'}, \texttt{eL}_1)$ ...	$\texttt{aL}_{31} = ((\texttt{tc\_label}), \texttt{c\_label}, 1, \texttt{'tc\_label'}, \texttt{eL}_{11})$ $\texttt{aL}_{32} = ((\texttt{rate}), \texttt{relevance}, 1, \texttt{'rate/100'}, \texttt{eL}_{11})$ ...

4.5.2.1 Appariement des étapes terminologique et termino-ontologique.

En considérant les étapes terminologique et termino-ontologique au travers de leur modèle respectif, le mapping présenté dans le tableau 4.4 peut être défini de la manière suivante :

- **Création des mLinks.** L'instruction SQL $\texttt{S}_1$ du tableau 4.5 permet d'insérer, dans la table *mLink* (Cf. figure 4.4), un tuple représentant un appariement du modèle terminologique avec le modèle termino-ontologique.

- **Création des eLinks.** L'instruction SQL $\texttt{S}_3$ du tableau 4.6 permet d'insérer, dans la table *eLink* (Cf. figure 4.4), un tuple représentant l'appariement de l'entité `Term` du modèle terminologique avec l'entité `TerminoConcept` du modèle termino-ontologique. De la même manière, un tuple est créé pour exprimer un appariement de l'entité `TermRelation` du modèle terminologique avec l'entité `TerminoConceptRelation` du modèle termino-ontologique.

- **Création des aLinks.** L'instruction SQL $\texttt{S}_5$ du tableau 4.7 insère, dans la table *aLink* (Cf. figure 4.4), un tuple exprimant que, lors de la transformation d'une instance de l'entité `Term` du modèle terminologique en une instance de l'entité `TerminoConcept` du modèle termino-ontologique, le *libellé* (`tc_label`) du *terminoconcept* créé est identique à celui du *terme* (`t_label`). De la même manière, un tuple est créé pour exprimer que le rendement (`rate`) d'une instance de l'en-

tité TerminoConcept est égal à la fréquence (frequency) du terme additionnée au nombre de variante du terme (variant_number).

Tableau 4.5 – Création des mLinks

```
S1
INSERT INTO mLink (source, target, confidence)
VALUES(Terminology, TerminoOntology, 0.9);

S2
INSERT INTO mLink (source, target, confidence)
VALUES(TerminoOntology, Ontology, 1);
...
```

Tableau 4.6 – Création des eLinks

```
S3
INSERT INTO eLink (source, target, confidence, mL)
VALUES(Term, TerminoConcept, 0.8, 500);

S4
INSERT INTO eLink (source, target, confidence, mL)
VALUES(TerminoConcept, Class, 0.9, 501);
...
```

4.5.2.2 Appariement des étapes termino-ontologique et ontologique.

En considérant les étapes termino-ontologique et ontologique au travers de leur modèle respectif, le mapping présenté dans le tableau 4.4 peut être défini de la manière suivante :

- **Création des mLinks.** L'instruction SQL S_2 du tableau 4.5 permet d'insérer, dans la table *mLink* (Cf. figure 4.4), un tuple représentant un appariement du modèle termino-ontologique avec le modèle ontologique.

- **Création des eLinks.** Dans le contexte du *mLink* précédemment créé, un *eLink* est créé de l'entité TerminoConcept du modèle termino-ontologique vers l'entité Class du modèle ontologique, exprimant ainsi que les instances de l'entité TerminoConcept du modèle terminologique seront transformées en des instances de l'entité Class du modèle termino-ontologique. Ce *eLink* est créé à l'aide de l'instruction S_4 représenté dans le tableau 4.6. De la même manière, un tuple est créé pour exprimer un appariement de l'entité TerminoConceptRelation du modèle termino-ontologique avec l'entité Property du modèle ontologique.

- **Création des aLinks.** L'instruction SQL S_7 du tableau 4.7 insère, dans la table *aLink* (Cf. figure 4.4), un tuple exprimant que, lors de la transformation d'une instance de l'entité TerminoConcept

du modèle termino-ontologique en une instance de l'entité `Class` du modèle ontologique, le *libellé* (`c_label`) de *classe* créée est identique à celui du *terminoconcept* (`tc_label`). De la même manière, un tuple est créé pour exprimer que la pertinence (`relevance`) d'une instance de l'entité `Class` est égal au rendement (`rate`) du *terminoconcept* correspondant divisé par `100`.

Tableau 4.7 – Création des aLinks

```
S5
INSERT INTO aLink (source, target, confidence, expression, eL)
VALUES((t_label), tc_label, 1, "t_label", 600);

S6
INSERT INTO aLink (source, target, confidence, expression, eL)
VALUES((frequency,variant_number), rate,
1, "frequency+variant_number", 600);

S7
INSERT INTO aLink (source, target, confidence, expression, eL)
VALUES((tc_label), c_label, 1, "tc_label", 601);

S8
INSERT INTO aLink (source, target, confidence, expression, eL)
VALUES((rate), relevance, 1, "rate/100", 601);
...
```

4.6 Conclusion

Dans ce chapitre, nous avons présenté notre approche de persistance des mappings. En nous intéressant au cas particulier des bases de données dites à base de modèles, nous avons proposé une infrastructure extensible pour la gestion des mappings. Contrairement aux autres approches, nous avons proposé une approche générique permettant de représenter les constructeurs de mappings au sein d'un méta-méta-modèle. Ce méta-méta-modèle est utilisé, d'une part, pour faciliter la création et la modification de nouveaux constructeurs de mappings, et, d'autre part, pour la génération automatique de la structure physique de persistance des mappings.

Une fois que les modèles et les mappings sont créés, il serait intéressant, par exemple, d'exploiter ces mappings lors de l'interrogation des données. En effet, comme dans le processus de construction d'ontologies, les mappings sont utilisés pour apparier des modèles hétérogènes modélisant le même domaine, le processus de recherche d'information dans un modèle pourrait exploiter les mappings. Néanmoins, l'infrastructure de persistance que nous avons présentée dans ce chapitre n'est pas facile à manipuler en utilisant le langage classique de requête SQL. Par exemple, comme les mappings sont transitifs (par composition de mappings), on souhaiterait pouvoir exploiter cette capacité pour interroger transitivement les données. Nous montrons, dans le chapitre suivant, que l'écriture d'une telle requête peut être complexe, et nous proposons, en vue de simplifier cette écriture, un langage de requête s'appuyant sur notre structure de BDBM et facilitant la gestion des instances et des modèles appariés par des mappings.

Chapitre 5

MQL, un langage d'exploitation des mappings

Sommaire

Résumé. Dans le chapitre 3, nous avons présenté notre approche de représentation des modèles et des mappings. L'implémentation de cette représentation dans un environnement persistant de bases de données a été proposée dans le chapitre précédent. Nous avons pour cela étendu le modèle de données des BDBM de manière à supporter les mappings. Dans ce chapitre, nous nous intéressons à l'exploitation de ces mappings. Après avoir présenté la difficulté d'exploiter une BDBM avec le langage de requête SQL, nous présentons notre proposition de langage visant à simplifier l'exploitation des BDBM stockant des mappings. Ce langage offre, entre autre, des opérateurs exploitant la transitivité des mappings.

5.1 Introduction

Dans le chapitre précédent, nous avons proposé une implémentation en base de données de notre approche de définition des mappings telle que spécifiée dans le chapitre 3. Cette implémentation offre la possibilité de stocker les mappings dans les bases de données dites à base de modèles. Dans ce chapitre, nous nous intéressons à l'exploitation des mappings ainsi persistés. En effet, nous souhaitons, par exemple, pouvoir interroger les modèles en raisonnant sur les mappings existants entre ces modèles. Nous montrons, dans la section 5.2, la difficulté de le faire en utilisant le langage de requête SQL et dégageons quelques exigences pour un langage facilitant l'exploitation des mappings. En effet, compte tenu de sa structuration, une BDBM reste difficile à manipuler, car cela suppose la parfaite maîtrise de ses différentes parties (instance, schéma et méta-schéma). Nous souhaitons donc rendre transparent, autant que possible, la complexité de cette structuration. Nous nous intéressons, par exemple, à l'exploration du graphe de mappings et nous montrons que cette opération, lors de l'interrogation des données, par exemple, impose à l'utilisateur d'écrire des requêtes syntaxiquement complexes et utilisant plusieurs jointures. Pour résoudre ce problème, nous proposons d'équiper notre infrastructure de bases données d'un langage permettant de manipuler, d'une part, les modèles, les mappings et les données, et, d'autre part, de manipuler conjointement les données et les mappings (Cf. section 5.3). Un tel langage a donc pour objectif principal de simplifier, pour un utilisateur, le parcours du graphe de mappings. Nous présentons dans ce chapitre le langage MQL (Mapping Query Language [Téguiak et al., 2012a]) que nous proposons à cette fin. Ce langage s'appuie sur une sémantique algébrique décrite au sein d'une algèbre appelée *MapAlgebra* (Cf. section 5.4).

5.2 Exigences

Un langage d'exploitation des BDBM conforme à la description présentée dans le chapitre précédent (c'est-à-dire une BDBM capable de stocker à la fois les instances, les modèles, les méta-modèles et les mappings) obéit à un certain nombre d'exigences. [Wakeman and Jowett, 1993, OMG, 2003, Patrascoiu, 2004, Jean et al., 2006, Petrov and Nemes, 2008] ont identifié des exigences liées à l'exploitation des données et des méta-données (des modèles par exemple). Nous résumons ces propositions et les enrichissons en proposant des exigences propres à l'exploitation des mappings.

5.2.1 Exigences de méta-modélisation

On distingue deux principaux types d'exigences liées à l'exploitation de méta-modèles par des langages de requêtes.

1. *Traitement uniforme des données et des méta-données*. Le langage doit permettre aux utilisateurs de manipuler de la même manière, les données et les méta-données, car les méta-données ne sont rien d'autre que des données, mais d'un point de vue plus abstrait. Ainsi, les opérateurs du langage de requêtes doivent être, autant que possible, indépendants du niveau d'abstraction.
2. *Simplicité et expressivité*. La syntaxe du langage de requêtes doit être simple, de manière à faciliter sa prise en main tout en étant suffisamment expressive et la moins ambigüe possible. Ces

caractéristiques sous-entendent de préserver une certaine compatibilité avec le langage classique de requêtes SQL utilisé pour les bases de données relationnelles.

5.2.2 Exigences d'exploitation des mappings

Dans les bases de données relationnelles, la recherche d'informations s'effectue grâce aux requêtes de la forme `"SELECT ... FROM ... WHERE ..."`. Ce type de requête ne prend pas en compte la notion de mapping car elle n'exploite pas les éventuels mappings existants entre les modèles. Dans la pratique, l'utilisateur doit lui-même prendre en compte la notion de mapping dans l'écriture de ses requêtes, qui, du point de vue syntaxique peut être complexe à écrire. En guise d'illustration, considérons à nouveau l'exemple de construction d'ontologies selon la démarche proposée par la plate-forme DaFOE dans laquelle nous avons, d'une part, associé un modèle à chaque étape, et, d'autre part, créé des mappings entre les modèles. Supposons également que les instances des modèles ontologique, termino-ontologique et terminologique sont respectivement représentés par les tables 5.1, 5.2 et 5.3. L'exploitation des mappings se manifeste, par exemple, lorsqu'un utilisateur qui interroge la table 5.1, souhaite également, pour chaque donnée retournée, obtenir une traçe de la construction de cette donnée à partir de données issues des tables 5.2 et 5.3. Pour le calcul de ces traçes, nous avons identifié quatre principales exigences permettant d'exploiter les mappings pour répondre au besoin d'un utilisateur.

Tableau 5.1 – Modèle Ontologique.

Class		
c_label	relevance	isAbtract
mairie	0.1	true
mariage	0.3	false
mariage religieux	0.2	true
personne	0.1	false
voiture	0.01	false
...	...	...

Tableau 5.2 – Modèle TerminoOntologique.

TerminoConcept		
tc_label	rate	définition
mairie	10	...
mariage	30	...
mariage religieux	20	...
personne	10	...
...	...	...

5.2.2.1 Minimisation de l'expression des requêtes

Grâce aux mappings entre les modèles, un utilisateur qui recherche les *classes* du modèle ontologique peut vouloir, par la même occasion, récupérer le(s) *termino-concept(s)* du modèle termino-ontologique utilisé(s) pour construire ces classes ou encore le(s) *terme(s)* (dans le modèle terminologique) utilisé(s)

Tableau 5.3 – Modèle Terminologique.

Term		
t_label	frequency	variant_number
mairie	1	0
mariage	25	5
mariage religieux	20	0
personne	9	1
...	...	...

pour construire le(s) *termino-concept(s)*. En effet, considérons l'exemple suivant :

Exemple 5.2.2.1

Rechercher toutes les classes du modèle ontologique dont le facteur de pertinence (`relevance`*) est supérieur à* `0.1` *ainsi que les instances (dans les modèles sources appariés au modèle ontologique) utilisées pour construire ces classes.*

Si on suppose l'existence de deux mapings: un mapping entre le modèle terminologique (TM) et le modèle terminoontologique (TOM) et un autre entre le modèle termino-ontologique (TOM) et le modèle ontologique (OM) tels que représentés dans le tableau 5.4 , deux scénarios sont envisageables pour répondre à l'exemple précédent.

Tableau 5.4 – Exemple de mappings.

	TM → TOM	**TOM → OM**
mLink	mL_1= (TM, TOM, 0.9)	mL_2= (TOM, OM, 1)
eLink	eL_1= (Term, TerminoConcept , 0.8 , mL_1) eL_2= (TermRelation, TerminoConceptRelation , 0.9 , mL_1) ...	eL_{11}= (TerminoConcept, Class, 0.9, mL_2) eL_{12}= (TerminoConceptRelation, Property, 0.8, mL_2) ...
aLink	aL_{21}= ((t_label), tc_label, 1, 't_label', eL_1) aL_{22}= ((frequency, variant_number), rate, 1 , 'frequency + variant_number', eL_1) ...	aL_{31}= ((tc_label), c_label, 1, 'tc_label', eL_{11}) aL_{32}= ((rate), relevance, 1, 'rate/100', eL_{11}) ...

1. Si l'utilisateur connaît les mappings entre les modèles, c'est-à-dire s'il connaît le contenu du tableau 5.4, alors il peut directement écrire les requêtes présentées dans le tableau 5.5.

   Notons que les requêtes $\mathbf{R}_2$ et $\mathbf{R}_3$ représentent les traductions respectives de la requête $\mathbf{R}_1$ sur les modèles termino-ontologique et terminologique en utilisant les mappings du tableau 5.4. En effet, la requête $\mathbf{R}_1$ est utilisée pour produire la requête $\mathbf{R}_2$ dans laquelle l'entité `Class` a été remplacée par l'entité `TerminoConcept` et l'attribut `c_label` (respectivement `relevance`) de l'entité `Class` a été remplacé par l'attribut `tc_label` (respectivement `rate/100`) conformément au mapping entre les modèles termino-ontologique et ontologique. La requête $\mathbf{R}_2$ obtenue est à son tour utilisée pour produire la requête $\mathbf{R}_3$ en utilisant le mapping entre les modèles terminologique et termino-ontologique.

Tableau 5.5 – Requêtes d'interrogation des modèles mappés.

ID	Requêtes
$\mathbf{R}_1$	**SELECT** c_label, relevance **FROM** Class **WHERE** relevance ≥ 0.1
$\mathbf{R}_2$	**SELECT** tc_label, rate **FROM** TerminoConcept **WHERE** rate/100 ≥ 0.1
$\mathbf{R}_3$	**SELECT** t_label, frequency, variant_number **FROM** Term **WHERE** (frequency + variant_number)/100 ≥ 0.1

Dans cette traduction de requêtes, seul le prédicat de sélection portant sur les attributs est traduit en exploitant l'expression sémantique décrivant l'attribut cible d'un *aLink* en fonction des attributs sources. En effet, en faisant également la traduction sur les attributs de la clause SELECT, les résultats obtenus ne seraient pas l'image exacte des tuples de la base de données.

2. Si l'utilisateur ne connaît pas les mappings ou s'il n'est pas sûr qu'ils n'aient pas été modifiés (de nouveaux mappings peuvent être créés, tout comme des mappings existants peuvent avoir été supprimés ou modifiés comme c'est le cas dans les environnements *peer to peer* [Klampanos and Jose, 2003, Halevy et al., 2004]), l'utilisateur doit :
   - tout d'abord interroger le graphe de mappings pour obtenir les mappings ;
   - ensuite, à partir de ces mappings, écrire la ou les requête(s) appropriée(s).

Si le second scénario, assez générique, est mieux adaptée à la modification des mappings, il nécessite un accès, dans la BDBM, à l'espace dédié aussi bien au stockage des modèles et des mappings (partie schéma de la BDBM) qu'à l'espace dédié au stockage des méta-modèles (partie méta-schéma de la BDBM). Cet accès conjoint aux différentes parties de la BDBM rend plus complexe la requête qui, dès lors, nécessite plusieurs sous-requêtes ainsi qu'un langage hôte. Pour simplifier notre illustration, nous supposons que la recherche sera propagée uniquement sur le modèle termino-ontologique. Ainsi, comme l'utilisateur ne connaît pas forcément l'ensemble décrivant le mapping entre les modèles ontologique et termino-ontologique, il écrit dans un premier temps une série de requêtes dites de niveau schéma car elles interrogent la partie schéma de la BDBM pour extraire des mappings. En utilisant le *méta-modèle noyau* (Cf. section 3.6 du chapitre 3), cette série de requêtes peut être illustrée de la manière suivante.

$\mathbf{Q}_1$) *Recherche du modèle ontologique.*

```
SELECT M.oid
FROM Entity E, Model M
WHERE E.label= "Class" AND E.model= M.oid
```

$\mathbf{Q}_2$) *Recherche des mLinks dans lesquels le modèle ontologique joue le rôle de modèle cible.*

```
SELECT mLink.oid
FROM mLink
WHERE mLink.target in (Q1)
```

$\mathbf{Q}_3$) *Recherche des entités sources appariées avec l'entité Class.*

```
SELECT eLink.oid, eLink.source
FROM eLink,Entity E
WHERE eLink.mL in (Q2)
AND eLink.oid= E.oid AND E.label= "Class"
```

$\mathbf{Q}_4$) *Recherche des entités appariées ainsi que les attributs appariés.*

```
SELECT E.label, aLink.source, aLink.expression
FROM Entity E, Attribute A, aLink
WHERE (aLink.eL, E.oid) in (Q3)
AND aLink.target= A.oid AND A.dom= E.oid
```

La requête $\mathbf{Q}_4$ retourne toutes les informations nécessaires pour traduire la requête sur l'entité `Class` du modèle ontologique en une requête sur l'entité `TerminoConcept` du modèle termino-ontologique. L'exécution de cette requête produit le résultat du tableau 5.6 exploitable pour écrire, pour le modèle termino-ontologique, la requête $Q_{TerminoOntology}$ représentant la traduction de la requête $Q_{Ontology}$ initiale. Cependant, l'exploitation du tableau 5.6 nécessite d'associer au langage SQL, un langage hôte permettant de manipuler le procédural en vue de générer la requête $Q_{TerminoOntology}$.

$\mathbf{Q}_{TerminoOntology}$)

```
SELECT tc_label , rate
FROM TerminoConcept
WHERE rate/100 ≥ 0.1
```

Tableau 5.6 – Résulats de niveau mapping.

E.label	aLink.source	aLink.expression
TerminoConcept	(tc_label)	'tc_label'
TerminoConcept	(rate)	'rate/100'

5.2.2.2 Assistance à l'exploration du graphe de mappings

En analysant davantage la deuxième situation de l'exigence présentée dans la section 5.2.2.1, on peut se demander comment gérer la transitivité des mappings à l'aide du langage SQL. En d'autres termes, comment effectuer les opérations de composition de mappings? Cette question fait référence à la difficulté d'explorer le graphe de mappings. En effet, sachant par exemple que dans le graphe de mappings, l'entité `Class` est appariée à l'entité `TerminoConcept`, et que l'entité `TerminoConcept` est à son tour appariée

à l'entité `Term`, notre objectif consiste à gérer la propagation d'une requête portant sur les *classes* en une requête portant sur les *terminoconcepts*, et, ensuite en une requête portant sur les *termes*. Une politique de contrôle de la propagation transitive des requêtes induites (obtenues par traduction sur d'autres modèles) à travers le graphe de mappings est donc nécessaire.

5.2.2.3 Saturation mémoire

La contrainte de saturation mémoire s'intéresse à la définition d'un protocole de gestion des données permettant d'éviter de charger un gros volume de données en mémoire centrale. En effet, le graphe de mappings peut très vite devenir volumineux, et donc coûteux (en temps de réponse et en mémoire consommée) à parcourir. Ceci s'explique par le fait que, comme nous l'avons évoqué, une nouvelle étape, et par conséquent un nouveau modèle, peut être définie dans le processus de construction d'ontologies proposé par la plate-forme DaFOE.

Afin de simplifier la tâche de l'utilisateur, un langage de requête prenant en compte la notion de mapping et respectant les exigences précédentes est nécessaire.

De façon analogue à l'interrogation des données, les mappings peuvent également être exploités lors de l'insertion, la suppression ou la modification des données. Dans le cas de l'insertion des données, par exemple, le besoin consiste à définir des mécanismes permettant, lors d'une insertion des données dans un modèle, de propager cette insertion sur les modèles appariés avec ce modèle. On définit des besoins analogues lors d'une mise à jour ou d'une suppression de données. Intuitivement, nous pouvons affirmer que les réponses à ces exigences respectent des fonctionnalités similaires à celles évoquées pour l'interrogation des données.

5.2.2.4 Cohérence des instances

Dans le chapitre précédent (Cf. section 4.4), nous avons vu que la modification des instances d'une entité source et l'évolution des modèles et des mappings n'étaient pas sans conséquence sur la cohérence des instances de l'entité cible d'un appariement d'entités. Nous avons notamment précisé les adaptations à faire pour préserver la cohérence des données de l'entité cible comparitivement à celles de l'entité source. Dans la suite de ce document nous utilisons le terme *synchronisation d'entités* pour désigner ce processus d'adaptation.

Dans le cas d'une évolution du mapping par exemple, nous pouvons, de façon analogue à l'interrogation des données, traiter le problème de la synchronisation d'entités selon deux considérations.

1. Si l'utilisateur connaît les mappings, c'est-à-dire s'il connaît le contenu du tableau 5.4, alors il peut directement écrire les requêtes présentées dans le tableau 5.7 pour lesquelles nous avons supposé qu'un utilisateur a décidé de supprimer l'appariement existant entre les entités `Term` et `TerminoConcept` . La requête $\mathbf{Q}_2$ permet de supprimer tous les *terminoconcepts* créés à partir des *termes*. Or, si l'on décide de supprimer des terminoconcepts, il faudrait également, en vertu de l'appariement existant entre les entités `TerminoConcept` et `Class`, supprimer toutes les classes

créées à partir des *terminoconcepts* à supprimer en utilisant la requête $\mathbf{Q}_1$.

2. Par contre, si l'utilisateur ne connaît pas les mappings, il a tout d'abord besoin, comme pour l'interrogation des données, d'interroger le graphe de mappings pour obtenir les mappings, et ensuite, à partir de ces mappings, d'écrire la ou les requête(s) de synchronisation appropriée(s). Notons que l'utilisation d'un langage hôte est également nécessaire dans ce cas.

On constate donc que, aussi bien lorsque l'utilisateur connaît les mappings que lorsqu'il ne les connaît pas, la *synchronisation d'entités* est une tâche qui nécessite d'écrire des requêtes pouvant être syntaxiquement complexes si l'on prend en compte la totalité du graphe de mappings.

Tableau 5.7 – Requêtes de synchronisation avant la suppression d'un eLink.

ID	Requêtes

$\mathbf{Q}_1$	`DELETE FROM Class as C`
	`WHERE (C.c_label, C.relevance ) in`
	`(SELECT TC.tc_label, TC.rate/100 FROM TerminoConcept as TC)`

$\mathbf{Q}_2$	`DELETE FROM TerminoConcept as TC`
	`WHERE (TC.tc_label, TC.rate ) in`
	`(SELECT T.t_label, T.frequency + TC.variant_number FROM Term as T)`

Les exigences précédemment établies n'étant pas supportées par les langages d'exploitation de mappings que nous avons présentés dans le chapitre 2 (Cf. section 2.8), nous proposons un nouveau langage nommé MQL que nous présentons dans les sections suivantes.

5.3 Notre approche

Dans cette section, nous présentons le langage MQL que nous proposons pour l'exploitation des mappings dans les BDBM. Ce langage possède, d'une part, une architecture de méta-modélisation conforme à l'exigence de méta-modélisation que nous avons présentée, et, d'autre part, fournit des opérateurs pour l'exploitation des mappings. La grammaire complète du langage MQL est disponible en annexe (Cf. Annexe A).

5.3.1 Architecture de méta-modélisation

Dans le chapitre précédent, nous avons montré que les BDBM étaient mieux adaptées pour la persistance des modèles et des mappings. Nous nous sommes notamment appuyés sur l'architecture de la base de données OntoDB pour proposer une BDBM supportant la gestion des mappings. Nous avons également vu que la BDBM OntoDB est équipée du langage de requête OntoQL qui permet d'exploiter les parties méta-schéma, ontologie et instance de la BDBM OntoDB. Comme nous l'avons vu dans le chapitre 2 (Cf. section 2.8.2), de nombreuses solutions éprouvées pour les langages de type méta-modélisation existent dans la littérature. Pour les besoins de la plate-forme DaFOE, nous proposons le langage de requêtes MQL supportant les opérations de base proposées par les langages de type méta-modélisation. Dans cette section, nous ne détaillons pas les aspects liés à la méta-modélisation tels que

proposés par ces langages mais nous nous focalisons davantage sur les aspects liés à l'exploitation des mappings qui constituent l'originalité du langage MQL.

Contrairement aux SGBD où le méta-modèle est figé, MQL s'appuie sur le méta-méta-modèle des BDBM stockant le *méta-modèle noyau* utilisé pour représenter les modèles et les mappings. Le langage MQL fournit des opérateurs permettant d'instancier et d'interroger le *méta-modèle noyau*. Comme ce *méta-modèle noyau* n'est pas figé, les éléments de niveau méta-modèle ne doivent pas être utilisés comme des mots-clés du langage MQL. De ce fait, pour définir la syntaxe du langage MQL, nous avons choisi d'étendre la syntaxe du langage SQL de manière à supporter les éléments de notre méta-modèle. Le langage MQL est constitué de sous-ensembles fonctionnels dédiés à la manipulation des différentes parties de notre BDBM.

Afin d'uniformiser le traitement des données et des méta-données, le langage MQL propose une syntaxe commune pour la création des tables des parties schéma et instance. On parle alors de création de structure (d'où le mot clé `STRUCT`).

5.3.2 Gestion de la partie méta-schéma

Les requêtes de l'exemple 5.3.2.1 créent les concepts `Model` et `mLink` dans le *méta-modèle noyau*. Le concept `mLink` est créé avec deux attributs, `source` et `target`, permettant d'identifier les modèles à apparier. En effet, la sémantique du mot-clé `REF` est équivalente à la notion de clé étrangère telle que décrite dans le langage SQL. Notons cependant que, pour faciliter la lecture des requêtes, les noms (*"labels"*) des objets sont utilisés à la place des identifiants (`REF (Model)`) est, par exemple, utilisé à la place de `REF(4)`). Comme résultat de cet exemple, les instances 4 et 7 sont insérées dans la table `MetaEntity` (Cf. figure 5.1) et deux tables (`Model` et `mLink`) sont automatiquement créées dans la partie schéma de la BDBM.

Exemple 5.3.2.1

Construction du méta-modèle noyau

❶ *Création du concept "Model".*

```
CREATE STRUCT Model (oid Integer, label String).
```

❷ *Création d'un constructeur de mappings.*

```
CREATE STRUCT mLink (oid Integer, source REF(Model), target REF(Model),
confidence Integer).
```

Une fois la structure de persistance des modèles et des mappings créée, des modèles et des mappings peuvent être créés, modifiés ou supprimés en utilisant le sous-ensemble fonctionnel de MQL dédié à cet effet. La syntaxe retenue pour ce sous-ensemble fonctionnel est identique à celle du langage SQL de manière à respecter l'exigence de méta-modélisation.

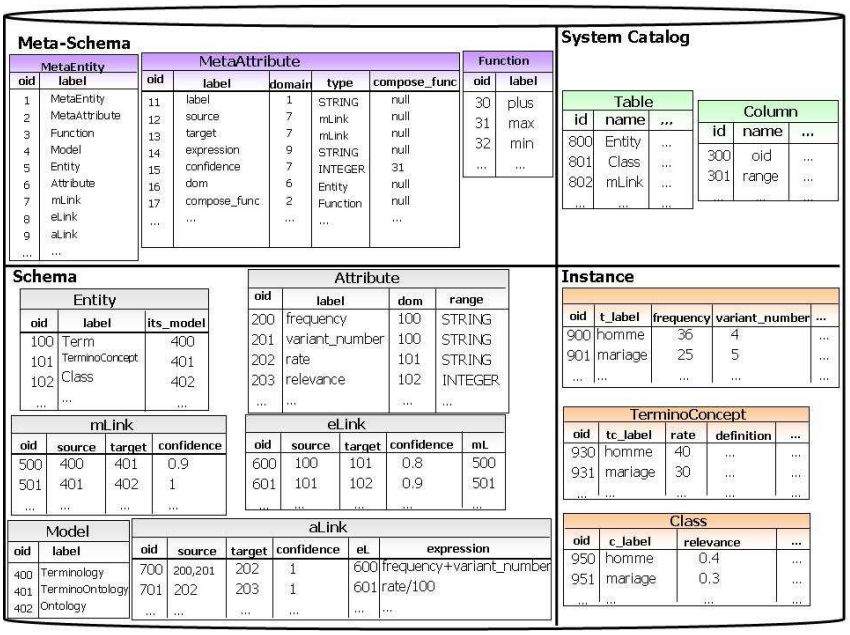

FIGURE 5.1 – Support des mappings dans la BDBM.

5.3.3 Gestion de la partie schéma

Le langage MQL offre un sous-ensemble fonctionnel compatible au langage SQL et permettant de manipuler le méta-modèle. L'exemple 5.3.3.2 illustre son utilisation pour la création d'un appariement de modèles.

Exemple 5.3.3.2

Création d'un appariement entre les modèles terminologique et termino-ontologique

```
INSERT INTO mLink(source, target, confidence)
VALUES ("Terminology", "TerminoOntology", 0.9).
```

De la même manière, nous définissons, pour le langage MQL, un sous-ensemble fonctionnel compatible au langage SQL et permettant d'interroger le méta-modèle. La syntaxe de ce sous-ensemble fonctionnel est donnée dans le tableau 5.8 et l'exemple 5.3.3.3 qui permet de rechercher un appariement de modèles illustre son utilisation.

Tableau 5.8 – Syntaxe d'interrogation du méta-modèle.

```
SELECT <attributes : VARCHAR [ ]>
FROM <entity_name : VARCHAR [ ]>
WHERE <criteria : PREDICATE [ ]>
...
```

Exemple 5.3.3.3

Rechercher tous les mappings dans lesquels le modèle terminologique est impliqué

```
SELECT label
FROM mLink
WHERE mLink.source = 'Terminology' OR mLink.target = 'Terminology'
```

5.3.4 Gestion de la partie instance

Le langage MQL permet également de manipuler des instances. Cette capacité à manipuler les instances est gérée par deux sous-ensembles fonctionnels. Le premier, nommé DQL (Data Query Language), fournit des opérateurs pour la recherche d'informations en interrogeant les modèles. Le second, nommé DML (Data Manipulation Language), est dédié à l'insertion, à la suppression et à la modification des instances. Une illustration du DML est présentée dans la section suivante, dans laquelle nous motivons le choix des extensions que nous avons apportées au langage SQL.

5.3.5 DQL : exploitation des mappings pour l'interrogation des données

Comme nous l'avons montré dans la section 5.2.2.1, le processus de génération de requêtes sur des modèles sources pourrait devenir complexe si l'on décide de prendre en compte l'ensemble des modèles présents dans le graphe de mappings. Dans ce cas, l'utilisateur aurait à sa charge la gestion et l'exploitation de la propriété de transitivité des mappings. Une solution intuitive à ce problème consiste, en général, à développer un traducteur de requêtes qui effectue une réécriture de requêtes. Ainsi, l'utilisateur écrit une requête ($Q_{Ontology}$ sur le modèle ontologique, par exemple), et le traducteur génère, par réécriture, des requêtes pour les différents modèles sources. Ce processus de génération de requêtes se fait en arrière-plan, c'est-à-dire de manière complètement transparente pour l'utilisateur. En d'autres termes, cette approche suppose que l'utilisateur ne dispose d'aucune information sur les mappings et ne peut donc pas influer le processus de réécriture et de génération de requêtes. Notre proposition, à travers le langage MQL, consiste à donner à un utilisateur disposant d'informations sur les mappings, la possibilité d'influer sur ce processus de réécriture/génération de requêtes en spécifiant notamment des préférences.

5.3.5.1 Notre proposition

L'un des principaux objectifs du langage MQL est de faciliter l'exploration du graphe de mappings. Nous avons donc introduit des clauses optionnelles permettant d'étendre le langage SQL et offrant une syntaxe compacte pour la propagation des requêtes d'un modèle à l'autre. Ainsi, comme solution aux exigences présentées dans la section 5.2 et pour s'affranchir des limitations ci-dessus évoquées, nous proposons d'étendre les requêtes SQL de type `"SELECT ... FROM ... WHERE ... "` avec des clauses exploitables par le traducteur de requêtes lors du processus de propagation. Le tableau 5.9 présente la syntaxe globale des requêtes MQL dédiées à l'interrogation des données incluant les contraintes sur les mappings tandis que des exemples illustrant l'utilisation de ces clauses sont présentés dans la section 5.3.5.2. Le rôle de chacune de ces clauses est :

Tableau 5.9 – Syntaxe du DQL.

```
SELECT <attributes : VARCHAR [ ]>
FROM <entity_name : VARCHAR [ ]>
WHERE <criteria : PREDICATE [ ]>
MATCH <mapped_models : VARCHAR [ ]>
FILTER <mapped_entities_name : VARCHAR [ ]>
CONFIDENCE <conf : INTEGER>
With closure
DEPTH <dep: INTEGER>
mWHERE <criteria : PREDICATE [ ]>
```

MATCH. Cette clause permet de spécifier les modèles sur lesquels on souhaite propager la requête MQL.

FILTER. Lors de la propagation de la requête MQL d'un modèle $\mathtt{m}_1$ sur un autre modèle $\mathtt{m}_2$, une entité du modèle $\mathtt{m}_1$ peut correspondre à plusieurs entités du modèle $\mathtt{m}_2$. Dans cette situation, on peut vouloir restreindre la propagation de manière à ce qu'elle ne concerne que certaines entités précises. La clause `FILTER` permet de créer une telle restriction.

CONFIDENCE. Comme un degré de confiance est souvent affecté aux mappings, cette clause permet de restreindre la propagation de la requête MQL uniquement à travers le sous-ensemble des appariements satisfaisant le degré de confiance spécifié. Lorsqu'elle est spécifiée, cette clause est utilisée comme le seuil minimal à respecter.

With closure. Cette clause, lorsqu'elle est spécifiée, ordonne la propagation transitive de la requête sur le graphe de mappings. De cette manière, on arrive à extraire les instances en raisonnant sur la propriété de transitivité des appariements.

DEPTH. Une requête MQL utilisant la clause `With closure` peut très rapidement créer une saturation mémoire ou encore produire un temps de réponse conséquent selon la taille du graphe de mappings. Cette taille n'est d'ailleurs pas forcément connue par l'utilisateur écrivant la requête. MQL adresse ce problème en proposant la clause `DEPTH` qui permet de spécifier la profondeur d'exploration du graphe de mappings. Par exemple, `"DEPTH 4"`, signifie que la requête MQL sera propagée de manière transitive sur au plus `4` modèles consécutifs i.e sur `4` arêtes consécutives du graphe.

mWHERE. A la différence de la clause classique `WHERE` du langage SQL, la clause `mWHERE` permet à l'utilisateur de spécifier des prédicats de recherche sur la partie schéma de la BDBM. En d'autres termes, la clause `mWhere` de MQL est comparable à la clause `WHERE`, mais dédiée à la manipulation de la partie schéma, en général, et pour la sélection des appariements, en particulier. De plus, toutes les clauses précédentes permettent à un utilisateur de spécifier des restrictions sur des attributs du *méta-modèle noyau*. Toutefois, compte tenu du fait que notre approche offre la possibilité d'étendre ce méta-modèle noyau (par création de nouveaux constructeurs de mappings ou par extension des constructeurs existants)

grâce au méta-schéma, la clause `mWHERE` offre également la possibilité de supporter de telles extensions dans l'écriture des requêtes MQL d'interrogation des données.

Plus concrètement, considérons le cas de la modélisation des mappings flous. L'approche classique de représentation de ces mappings flous consiste, pour chaque mapping, à affecter un degré de confiance compris entre 0 et 1. Supposons qu'un utilisateur qui souhaite modéliser autrement les mappings flous définisse pour chaque appariement, par exemple, une propriété de qualité pouvant prendre l'une des valeurs *"faible", "moyen", "bien", "excellent", etc.*. Dans ce contexte, chaque constructeur de mappings (*mLink, eLink, aLink*) sera étendu par l'ajout d'une nouvelle propriété (appelée *quality* par exemple) et pouvant prendre l'une des valeurs *"faible", "moyen", "bien", "excellent", etc.*. Une fois ces constructeurs de mappings modifiés, il est possible de les insérer dans une requête MQL en utilisant la clause `mWHERE`. Par exemple, la requête `SELECT ... mWHERE quality = "excellent"` sera propagée sur le graphe de mappings en utilisant uniquement les appariements vérifiant la contrainte `quality = "excellent"`.

5.3.5.2 MQL en action

Cette section présente l'utilisation du langage MQL comme solution de l'exemple 5.2.2.1 consistant à rechercher toutes les classes du modèle ontologique dont la pertinence (`"relevance"`) est supérieure à `0.1` ainsi que les instances (des autres modèles) qui ont permis de créer ces classes.

1. Appliquée au modèle ontologique, la requête $\mathbf{mQ}_1$ retourne (Cf. tableau 5.10) des instances extraites uniquement du modèle ontologique (aucune clause orientée mapping n'étant utilisée dans cette requête). En d'autres termes, cette requête n'est rien d'autre qu'une requête SQL classique. Notons que, pour faciliter la compréhension des résultats, chaque instance retournée par la requête est préfixée par le nom de l'entité à laquelle elle appartient.

Tableau 5.10 – Exemple requête MQL.

$\mathbf{mQ}_1$	**Résultats**

`SELECT c_label, relevance`
`FROM Class`
`WHERE relevance ≥ 0.1`
	Class(car, 0.1)
	Class(electric_motor, 0.4)
	...

2. Appliquée au modèle ontologique, la requête $\mathbf{mQ}_2$ retourne (Cf. tableau 5.11) des instances extraites des modèles ontologique et termino-ontologique (la clause `MATCH` de cette requête étant spécifiée sur le modèle termino-ontologique).

Tableau 5.11 – Exemple requête MQL.

$\mathbf{mQ}_2$	**Résultats**

`SELECT c_label, relevance`
`FROM Class`
`WHERE relevance ≥ 0.1`
`MATCH TerminoOntology`
	Class(car, 0.1)
	Class(electric_motor, 0.4)
	TerminoConcept(motorcycle, 8)
	...

3. Bien que la clause `MATCH` de la requête $\mathbf{mQ}_3$ soit spécifiée par le symbole `"*"`, cette requête, appliquée au modèle ontologique, retourne (Cf. tableau 5.12) des instances extraites uniquement

du modèle termino-ontologique. En effet, compte tenu la clause `DEPTH 1` qui limite l'exploration à une unique propagation transitive, seul le modèle termino-ontologique est accessible à partir du modèle ontologique.

Tableau 5.12 – Exemple requête MQL.

$\mathbf{mQ}_3$	**Résultats**

`SELECT c_label, relevance`
`FROM Class`
`WHERE relevance ≥ 0.1`
`MATCH` `*`
`FILTER` `*`
`With closure`
`DEPTH` `1`
	Class(car, 0.1)
	Class(electric_motor, 0.4)
	TerminoConcept(motorcycle, 8)
	...

4. Appliquée au modèle ontologique, la requête $\mathbf{mQ}_4$ retourne (Cf. tableau 5.13) des instances extraites des modèles ontologique, termino-ontologique et terminologique. En d'autres termes, grâce au symbole **`"*"`** de la clause `MATCH` et en l'absence de limitation de profondeur par la clause `DEPTH`, la requête $\mathbf{mQ}_4$ est propagée sur tous les modèles accessibles par transitivité à partir du modèle ontologique.

Tableau 5.13 – Exemple requête MQL.

$\mathbf{mQ}_4$	**Résultats**

`SELECT c_label, relevance`
`FROM Class`
`WHERE relevance ≥ 0.1`
`MATCH` `*`
`FILTER` `*`
`With closure`
	Class(car, 0.1)
	Class(electric_motor, 0.4)
	TerminoConcept(motorcycle, 8)
	Term(bicycle, 28,2)
	...

5.3.6 DML : exploitation des mappings dans la manipulation des données

En utilisant les mêmes considérations que pour le DQL du langage MQL, nous proposons pour le langage MQL, un sous-ensemble fonctionnel pour la manipulation des instances permettant de réaliser l'insertion, la suppression et la modification des instances en exploitant les mappings. La définition de ce sous-ensemble fonctionnel consiste également à étendre les opérateurs équivalents du langage SQL. En effet, chaque opérateur du langage SQL (`INSERT, UPDATE et DELETE`) permettant la manipulation des instances est étendu avec les mêmes clauses utilisées pour le DQL (`MATCH, FILTER`, etc.). De cette manière, en utilisant les clauses optionnelles dédiées aux mappings, l'insertion (respectivement la modification ou la suppression) d'une instance d'un modèle source provoque immédiatement l'insertion (respectivement la modification ou la suppression), dans les modèles cibles appariés au modèle source, d'instance(s) (calculée(s) en utilisant le graphe de mappings). Les tableaux 5.14, 5.15 et 5.16 résument la syntaxe du DML du langage MQL.

Tableau 5.14 – Syntaxe du DML pour l'insertion des données.

```
INSERT INTO <entity_name : VARCHAR> <attributes: VARCHAR [ ]>
VALUES <attributes_values : OBJECT [ ]>
MATCH <mapped_models : VARCHAR [ ]>
FILTER <mapped_entities_name : VARCHAR [ ]>
CONFIDENCE <conf : INTEGER>
With closure
DEPTH <dep : INTEGER>
mWHERE <criteria : PREDICATE [ ]>
```

Tableau 5.15 – Syntaxe du DML pour la modification des données.

```
UPDATE <entity_name : VARCHAR>
SET <attributes_name : VARCHAR> = <attributes_value: OBJECT>
WHERE <criteria: PREDICATE[ ]>
MATCH <mapped_models : VARCHAR [ ]>
FILTER <mapped_entities_name : VARCHAR [ ]>
CONFIDENCE <conf : INTEGER>
With closure
DEPTH <dep : INTEGER>
mWHERE <criteria : PREDICATE [ ]>
```

Tableau 5.16 – Syntaxe du DML pour la suppression des données.

```
DELETE FROM <entity_name : VARCHAR>
WHERE <criteria: PREDICATE[ ]>
MATCH <mapped_models : VARCHAR [ ]>
FILTER <mapped_entities_name : VARCHAR [ ]>
CONFIDENCE <conf : INTEGER>
With closure
DEPTH <dep : INTEGER>
mWHERE <criteria : PREDICATE [ ]>
```

5.3.7 MQL et la synchronisation des entités

Dans le chapitre précédent (Cf. section 4.4), nous avons vu que synchronisation des entités de deux modèles peut nécessiter d'écrire des requêtes syntaxiquement complexes. Pour simplifier l'écriture d'une telle requête de synchronisation, le langage MQL propose une instruction de haut niveau permettant de réaliser la synchronisation. Cette instruction, établie entre deux entités pour lesquelles il existe un *eLink*, permet d'adapter/alimenter les données de l'entité cible du *eLink* en fonction des données de l'entité source du *eLink*. Pour affiner le processus de synchronisation, l'utilisateur peut éventuellement :

- spécifier des *aLinks* à utiliser pour la synchronisation ;
- spécifier le mode de synchronisation, qui peut prendre l'une des valeurs `onDelete`, `onUpdate` ou `onCreate` indiquant s'il s'agit respectivement d'une synchronisation liée à la suppression, la modification ou la création des *aLinks* spécifiés ;
- utiliser la clause `With Closure` pour préciser si la synchronisation doit se propager transitivement sur le graphe de mappings ;

– utiliser la clause DEPTH pour préciser la profondeur de la propagation transitive de la synchronisation.

Le tableau 5.17 illustre, pour la plate-forme DaFOE, la synchronisation des entités TerminoConcept et Term après la création des différents appariements entre leurs attributs. Le symbole "*" signifie que tous les *aLinks* créés entre les attributs seront utilisés. Dans cette illustration, l'utilisation de la clause With closure ordonne une synchronisation de l'entité Class du modèle ontologique.

Tableau 5.17 – Exemple de synchronisation de deux entités.

```
SYNCHRONIZE TerminoConcept, Term
WHERE Term.frequency ≥ 10
USING *
onCreate
With closure
DEPTH 2
```

5.4 Une algèbre pour le langage MQL

Dans cette section, nous présentons une représentation formelle du langage MQL. Cette représentation formelle s'appuie sur l'algèbre *MapAlgebra* qui est une adaptation de l'algèbre relationnelle pour la prise en compte des mappings. Nous dégageons également un ensemble de contraintes à vérifier et nous décrivons les algorithmes utilisés pour le traitement des requêtes MQL.

5.4.1 Algèbre relationnelle

L'algèbre relationnelle est le support mathématique sur lequel repose le modèle relationnel proposé en 1970 par Edgar Frank Codd [Codd, 1970]. Cette algèbre comporte un ensemble d'opérations qui permettent de garantir sa complétude.

5.4.1.1 Concepts

L'algèbre relationnelle manipule les principaux concepts suivants :
– des relations, représentées au sein d'un ensemble $\mathcal{R}$;
– des attributs, associés aux relations et représentés par un ensemble $\mathcal{A}$ permettent de décrire en intention ces relations ;
– des tuples ou instances de relations, représentés par un ensemble $\mathcal{T}$;
– des domaines primitifs (Integer, String, Boolean, etc.) représenté par un ensemble $\mathcal{D}$ et permettant de typer les attributs des entités ;
– etc..

Remarque. Soient $t \in \mathcal{T}$ un tuple et $\mathtt{R} \in \mathcal{R}$ une entité, par simplification d'écriture, nous utilisons la notation $\mathtt{t} \in \mathtt{R}$ pour exprimer le fait que l'instance t est associée (ou appartient) à l'entité R.

5.4.1.2 Opérations

L'algèbre relationnelle possède de nombreuses opérations. Nous présentons, de façon non exhaustive quelques opérations de cette algèbre.

1. **La Sélection.**
   Lors de la sélection des instances, on a en général besoin de spécifier un prédicat de sélection permettant de filtrer des instances vérifiant une certaine condition. On dit dans ce cas que les instances sélectionnées satisfont le prédicat ou encore que l'évaluation du prédicat, en utilisant ces instances, est vraie. Un prédicat comporte un ou plusieurs attributs. L'expression "`relevance` $\geq$ `0.1`" est un exemple de prédicat de sélection sur l'entité `Class` du modèle ontologique où `relevance` est un attribut.

   Définition 5.4.1

   Représentation d'un prédicat

   Si $\mathcal{P}$ désigne l'ensemble des prédicats, un prédicat `Pred` $\in \mathcal{P}$ *est défini par :*

   $$\begin{array}{rccl} \texttt{Pred}: & \mathcal{I} & \longrightarrow & \mathcal{BOOLEAN} \\ & \texttt{t} & \longmapsto & \texttt{Pred(t)} = \begin{cases} \texttt{TRUE} \textit{ si l'instance } \texttt{t} \textit{ satisfait } \texttt{Pred} \\ \texttt{FALSE} \textit{ sinon} \end{cases} \end{array}$$

   Ainsi, la sélection des tuples d'une relation `R` suivant un prédicat `Pred` peut être formellement définie par : $\texttt{SELECT(R, Pred)} = \{\texttt{t} \in \texttt{R} \mid \texttt{Pred(t)} = \texttt{true}\}$.

2. **La Projection.**
   De manière formelle, la projection d'une relation `R` suivant une liste d'attributs `A` peut être définie par : $\texttt{PROJECT(R, A)} = \{\langle \texttt{t.A} \rangle \mid \texttt{t} \in \texttt{R}\}$ où la notation $\langle \texttt{t.A} \rangle$ symbolise, pour un tuple `t`, sa restriction sur l'ensemble des attributs de `A`.

3. **Le Produit Cartésien.**
   De manière formelle, le produit cartésien de deux relations `R` et `S` peut être défini par :
   $\texttt{PRODUCT(R, S)} = \{(\texttt{t}_1, \texttt{t}_2) \in \texttt{R} \times \texttt{S}\}$

4. **L'Intersection.**
   De manière formelle, l'intersection de deux relations `R` et `S` peut être définie par :
   $\texttt{INTERSECT(R, S)} = \{\texttt{t} \mid \texttt{t} \in \texttt{R} \ \wedge \ \texttt{t} \in \texttt{S}\}$

5. **L'Union.**
   De manière formelle, l'union de deux relations `R` et `S` peut être définie par :
   $\texttt{UNION(R, S)} = \{\texttt{t} \mid \texttt{t} \in \texttt{R} \ \vee \ \texttt{t} \in \texttt{S}\}$

5.4.2 MapAlgebra : une adaptation de l'algèbre relationnelle pour la prise en compte des mappings

Dans le chapitre 3, nous avons présenté les représentations des modèles et des mappings. Ces représentations ont été regroupées sous le nom de *méta-modèle noyau* ou *core metamodel*. Dans cette section, nous présentons les concepts et les opérations de *MapAlgebra*, l'algèbre que nous avons définie pour le langage MQL.

5.4.2.1 Concepts de MapAlgebra

Les concepts de *MapAlgebra* permettent notamment de :

- caractériser l'ensemble constitué de tous les modèles à interconnecter par des mappings ;
- caractériser un modèle particulier ;
- décrire la représentation des mappings et leur exploitation.

Ainsi, en réutilisant les représentations des modèles et des mappings proposées dans le chapitre 3 (Cf. sections 3.4.2 et 3.5), nous définissons les concepts de *MapAlgebra* de la manière suivante.

Définition 5.4.2

Caractérisation de l'ensemble des modèles

Considérons les ensembles et les applications suivants :

- $\mathcal{M}$ *est l'ensemble des modèles ;*
- $\mathcal{E}$ *est l'ensemble des entités des modèles ;*
- $\mathcal{A}$ *représente l'ensemble des attributs utilisés pour décrire les entités des modèles ;*
- $\mathcal{T}$ *est l'ensemble des types primitifs (Integer, String, Boolean, etc.) utilisés pour typer les attributs des entités ;*
- $\mathcal{I}$ *représente l'ensemble des instances des entités de tous les modèles ;*

- *dom* : $\mathcal{A} \rightarrow \mathcal{E}$ *définit le domaine d'un attribut ;*
- *range* : $\mathcal{A} \rightarrow \mathcal{E} \cup \mathcal{T}$ *définit le codomaine d'un attribut ;*
- *its_model* : $\mathcal{E} \rightarrow \mathcal{M}$ *retourne le contexte de l'entité, c'est-à-dire l'unique modèle associé à l'entité ;*
- *its_entity* : $\mathcal{I} \rightarrow \mathcal{E}$ *retourne le contexte de l'instance, c'est-à-dire l'unique entité associée à l'instance. A l'inverse,* $\forall(\mathtt{i},\mathtt{e}) \in \mathcal{I} \times \mathcal{E}$ *nous utilisons la notation "*$\mathtt{i} \in \mathtt{e}$*" pour exprimer que* $\mathtt{i}$ *est une instance de* $\mathtt{e}$.

Définition 5.4.3

Caractérisation d'un modèle

Un modèle $\mathtt{m} = (E, A, T, I, dom, range, its\_model, its\_entity) \in \mathcal{M}$ *est défini par :*

- $E \subseteq \mathcal{E}$ *est l'ensemble des entités du modèle* $\mathtt{m}$ *i.e* $\mathtt{E} = \{\mathtt{e} \in \mathcal{E} \mid \mathtt{its\_model}(\mathtt{e}) = \mathtt{m}\}$;
- $A \subseteq \mathcal{A}$ *représente l'ensemble des attributs utilisés pour décrire les entités du modèle* $\mathtt{m}$ *i.e* $\mathtt{A} = \{\mathtt{a} \in \mathcal{A} \mid \mathtt{its\_model}(\mathtt{dom}(\mathtt{a})) = \mathtt{m}\}$;

- $T \subseteq \mathcal{T}$ *est l'ensemble des types primitifs utilisés pour typer les attributs des entités du modèle* $\mathtt{m}$;
- $I \subseteq \mathcal{I}$ *représente l'ensemble des instances des entités du modèle* $\mathtt{m}$ *i.e* $\mathtt{I} = \{\mathtt{i} \in \mathcal{I} \mid \mathtt{its\_model(its\_entity(i))} = \mathtt{m}\}$;
- $dom : A \rightarrow E$;
- $range : A \rightarrow E \cup T$;
- $its\_entity : I \rightarrow E$.

Pour la représentation et la manipulation des mappings, *MapAlgebra* définit les ensembles et les applications suivants.

Définition 5.4.4

Représentation et manipulation des mappings

- $\mathcal{F}_{\mathtt{sem}}$ *est un ensemble d'expressions utilisant des variables référant un attribut de* $\mathcal{A}$;

- $\mathcal{G}_{\mathtt{m}} = (\mathcal{N}_{\mathtt{m}}, \mathcal{L}_{\mathtt{m}})$ *représente le graphe d'appariement de modèles où :*
  - ❶ $\mathcal{N}_{\mathtt{m}} \subseteq \mathcal{M}$ *représente l'ensemble des nœuds du graphe d'appariement de modèles ;*
  - ❷ $\mathcal{L}_{\mathtt{m}} = \{(\mathtt{m_s}, \mathtt{m_t}, \alpha_{\mathtt{m}}) \in \mathcal{N}_{\mathtt{m}} \times \mathcal{N}_{\mathtt{m}} \times [\mathtt{0}, \mathtt{1}] \mid \mathtt{m_s} \neq \mathtt{m_t}\}$ *représente l'ensemble des appariements entre un modèle source* ($\mathtt{m_s}$) *et un modèle cible* ($\mathtt{m_t}$) *où* $\alpha_{\mathtt{m}}$ *est le degré de confiance de l'appariement.*
- $\mathcal{G}_{\mathtt{e}} = (\mathcal{N}_{\mathtt{e}}, \mathcal{L}_{\mathtt{e}})$ *représente le graphe d'appariement d'entités où :*
  - ❶ $\mathcal{N}_{\mathtt{e}} \subseteq \mathcal{E}$ *représente l'ensemble des noeuds du graphe d'appariement d'entités ;*
  - ❷ $\mathcal{L}_{\mathtt{e}} = \{(\mathtt{e_s}, \mathtt{e_t}, \alpha_{\mathtt{e}}, \mathtt{mL}) \in \mathcal{N}_{\mathtt{e}} \times \mathcal{N}_{\mathtt{e}} \times [\mathtt{0}, \mathtt{1}] \times \mathcal{L}_{\mathtt{m}}\}$ *représente l'ensemble des appariements entre une entité source* ($\mathtt{e_s}$) *et une entité cible* ($\mathtt{e_t}$) *avec un degré de confiance* $\alpha_{\mathtt{e}}$ *et associé à un mLink* $\mathtt{mL} \in \mathcal{L}_{\mathtt{m}}$ *qui représente son contexte de création.*
- $\mathcal{G}_{\mathtt{a}} = (\mathcal{N}_{\mathtt{a}}, \mathcal{L}_{\mathtt{a}})$ *représente le graphe d'appariement des attributs où :*
  - ❶ $\mathcal{N}_{\mathtt{a}} \subseteq \mathcal{A}$ *représente l'ensemble des noeuds du graphe d'appariement d'attributs ;*
  - ❷ $\mathcal{L}_{\mathtt{a}} = \left\{(\mathtt{A_s}, \mathtt{a_t}, \alpha_{\mathtt{a}}, \varphi, \mathtt{eL}) \in \mathcal{N}_{\mathtt{a}}^{\mathtt{k}} \times \mathcal{N}_{\mathtt{a}} \times [\mathtt{0}, \mathtt{1}] \times \mathcal{F}_{\mathtt{sem}} \times \mathcal{L}_{\mathtt{e}}\right\}$ *représente les noeuds d'appariement d'un ensemble d'attributs* $\mathtt{A_s} = (\mathtt{a_{s1}}, \mathtt{a_{s2}}, ..., \mathtt{a_{sk}})$ *avec un attribut cible* ($\mathtt{a_t}$) *avec le degré de confiance* $\alpha_{\mathtt{a}}$ *et associé à un eLink* $\mathtt{eL} \in \mathcal{L}_{\mathtt{e}}$ *qui représente son contexte de création.* $\varphi \in \mathcal{F}_{\mathtt{sem}}$ *est une expression décrivant* $\mathtt{a_t}$ *en fonction des* $\mathtt{a_{si}}$ $(\mathtt{1} \leq \mathtt{i} \leq \mathtt{k})$. *Dans un appariement d'attributs, tous les* $\mathtt{a_{si}}$ *sont associés à la même entité source.*

- $\mathtt{mapM} : \mathcal{M} \rightarrow 2^{\mathcal{M}}$ *retourne l'ensemble des modèles sources appariés avec un modèle cible donné ;*
- $\mathtt{mapE} : \mathcal{M} \times \mathcal{E} \rightarrow 2^{\mathcal{E}}$ *retourne, pour un modèle source donné, l'ensemble des entités sources appariées avec une entité cible donnée ;*
- $\mathtt{mapA} : \mathcal{E} \times \mathcal{A} \rightarrow 2^{\mathcal{A}}$ *retourne, pour une entité source donnée, l'ensemble des attributs sources appariés avec un attribut cible donné ;*
- $\mathtt{mapA}\varepsilon : \mathcal{E} \times \mathcal{A} \rightarrow \mathcal{F}_{\mathtt{sem}}$ *retourne, pour une entité source donnée, l'expression correspondant aux attributs sources appariés avec un attribut cible donné ;*

Les applications mapM, mapE, mapA, mapAε constituent des composantes fondamentales pour l'exploitation de *MapAlgebra*. Elles sont formellement définies de la manière suivante :

1. **mapM.**
   Soit $\mathtt{m}' \in \mathcal{M}$, $\mathtt{mapM}(\mathtt{m}') = \{\mathtt{m} \in \mathcal{M} \mid \exists \mathtt{mL} = (\mathtt{m}, \mathtt{m}', \alpha) \in \mathcal{L}_{\mathtt{m}}\}$.

   Dans le cas de la plate-forme DaFOE, on a par exemple : mapM(Ontology_Model(OM))={Termino-Ontology_Model(TOM), Terminology_Model(TM)}. En effet,

   ❶ $\mathtt{TOM} \in \mathtt{mapM(OM)}$ car $\exists \mathtt{mL}_1 = (\mathtt{TOM}, \mathtt{OM}, 1) \in \mathcal{L}_{\mathtt{m}}$ (*mLink* direct entre le modèle termino-ontologique et le modèle ontologique)

   ❷ $\mathtt{TM} \in \mathtt{mapM(OM)}$ car $\exists \mathtt{mL}_3 = (\mathtt{TM}, \mathtt{OM}, 1) \in \mathcal{L}_{\mathtt{m}}$ (*mLink* obtenu par composition des *mLinks* $\mathtt{mL}_1 = (\mathtt{TOM}, \mathtt{OM}, 1)$ et $\mathtt{mL}_2 = (\mathtt{TM}, \mathtt{TOM}, \mathtt{0.9})$).

2. **mapE.**
   Soient $\mathtt{m} \in \mathcal{M}$ et $\mathtt{e}' \in \mathcal{E}$, $\mathtt{mapE}(\mathtt{m}, \mathtt{e}') = \{\mathtt{e} \in \mathcal{E} \mid (\exists \mathtt{mL} = (\mathtt{m}, \mathtt{its\_model}(\mathtt{e}'), \alpha) \in \mathcal{L}_{\mathtt{m}}) \wedge (\exists \mathtt{eL} = (\mathtt{e}, \mathtt{e}', \alpha, \mathtt{mL}) \in \mathcal{L}_{\mathtt{e}}) \wedge (\mathtt{its\_model}(\mathtt{e}') \neq \mathtt{m})\}$.

   Dans le cas de la plate-forme DaFOE, on a par exemple :
   mapE(TerminoOntology_Model,Class) = {TerminoConcept}.

3. **mapA.**
   Soient $\mathtt{e} \in \mathcal{E}$ et $\mathtt{a}' \in \mathcal{A}$, , posons $\mathtt{m} = \mathtt{its\_model}(\mathtt{e})$ et $\mathtt{m}' = \mathtt{its\_model}(\mathtt{dom}(\mathtt{a}')$.
   $\mathtt{mapA}(\mathtt{e}, \mathtt{a}') = \{\mathtt{a}' \in \mathtt{A_s} \mid (\exists \mathtt{mL} = (\mathtt{m}, \mathtt{m}'), \alpha) \in \mathcal{L}_{\mathtt{m}}) \wedge (\exists \mathtt{eL} = (\mathtt{e}, \mathtt{dom}(\mathtt{a}'), \alpha, \mathtt{mL}) \in \mathcal{L}_{\mathtt{e}}) \wedge (\exists \mathtt{aL} = (\mathtt{A_s}, \mathtt{a}', \alpha, \mathtt{eL}, \varphi) \in \mathcal{L}_{\mathtt{a}}) \wedge (\mathtt{m} \neq \mathtt{m}') \wedge (\mathtt{e} \neq \mathtt{dom}(\mathtt{a}'))\}$.

   Dans le cas de la plate-forme DaFOE, on a par exemple :
   mapA(TerminoConcept,relevance) = {rate}.

4. **mapAε.**
   Soient $\mathtt{e} \in \mathcal{E}$ et $\mathtt{a}' \in \mathcal{A}$, posons $\mathtt{m} = \mathtt{its\_model}(\mathtt{e})$ et $\mathtt{m}' = \mathtt{its\_model}(\mathtt{dom}(\mathtt{a}')$. On a:
   $\mathtt{mapA}\varepsilon(\mathtt{e}, \mathtt{a}') = \{\mathtt{f} \in \mathcal{F}_{\mathtt{sem}} \mid (\exists \mathtt{mL} = (\mathtt{m}, \mathtt{m}'), \alpha) \in \mathcal{L}_{\mathtt{m}}) \wedge (\exists \mathtt{eL} = (\mathtt{e}, \mathtt{dom}(\mathtt{a}'), \alpha, \mathtt{mL}) \in \mathcal{L}_{\mathtt{e}}) \wedge (\exists \mathtt{aL} = (\mathtt{A_s}, \mathtt{a}', \alpha, \mathtt{eL}, \mathtt{f}) \in \mathcal{L}_{\mathtt{a}}) \wedge (\mathtt{m} \neq \mathtt{m}') \wedge (\mathtt{e} \neq \mathtt{dom}(\mathtt{a}'))\}$.

   Dans le cas de la plate-forme DaFOE, nous avons par exemple :
   mapAε(Term,relevance) = {'(frequency + variant_number)/100'}. En effet,
   $\exists$ aL_3= ((frequency, variant_number), rate, 1 , '(frequency + variant_number)/100', eL_{13})$\in$ $\mathcal{L}_a$ obtenu par composition des *aLinks* directs suivants :

   ❶ aL_2= ((rate), relevance, 1, 'rate/100', eL_{23}) qui permet de construire
   mapAε(TerminoConcept,relevance) = {'rate/100'};

❷ aL_1= ((frequency, variant_number), rate, 1 , 'frequency + variant_number', eL_{12}) qui permet de construire `mapAε(Term,rate) = {'frequency + variant_number'}`.

Théorème 5.4.2.1

S'il existe un appariement d'un attribut cible ($\mathtt{a}' \in \mathcal{A}$) avec un ensemble d'attributs sources (ces attributs sources appartiennent à une même entité $\mathtt{e}$), alors l'expression $\mathtt{f} \in \mathcal{F}_{\mathtt{sem}}$ de cet appariement est unique (i.e $\|\mathtt{mapA}\varepsilon(\mathtt{e},\mathtt{a}')\| = 1$).

Preuve.

Supposons qu'il existe un appariement d'un attribut cible ($\mathtt{a}' \in \mathcal{A}$) avec un ensemble d'attributs sources appartenant à une même entité $\mathtt{e} \in \mathcal{E}$ (avec $\mathtt{e} \neq \mathtt{dom}(\mathtt{a}')$). Par l'absurde, supposons que cet appariement admet deux expressions $\mathtt{f}_1$ et $\mathtt{f}_2$ appartenant à $\mathcal{F}_{\mathtt{sem}}$ et montrons que $\mathtt{f}_1 = \mathtt{f}_2$.

Posons $\mathtt{m} = \mathtt{its\_model}(\mathtt{e})$, $\mathtt{m}' = \mathtt{its\_model}(\mathtt{dom}(\mathtt{a}'))$ et $\mathtt{e}' = \mathtt{dom}(\mathtt{a}')$.

$$\mathtt{f}_1 \in \mathtt{mapA}\varepsilon(\mathtt{e},\mathtt{a}') \Leftrightarrow \begin{cases} \exists\, \mathtt{l}_{\mathtt{m}_1} = (\mathtt{m},\mathtt{m}',\alpha_{\mathtt{m}_1}) \in \mathcal{L}_{\mathtt{m}} & \textbf{(i)} \\ \exists\, \mathtt{l}_{\mathtt{e}_1} = (\mathtt{e},\mathtt{e}',\alpha_{\mathtt{e}_1},\mathtt{l}_{\mathtt{m}_1}) \in \mathcal{L}_{\mathtt{e}} & \textbf{(ii)} \\ \exists\, \mathtt{l}_{\mathtt{a}_1} = (\mathtt{A}_{\mathtt{s}},\mathtt{a}',\alpha_{\mathtt{a}_1},\mathtt{l}_{\mathtt{e}_1},\mathtt{f}_1) \in \mathcal{L}_{\mathtt{a}} & \textbf{(iii)} \end{cases}$$

$$\mathtt{f}_2 \in \mathtt{mapA}\varepsilon(\mathtt{e},\mathtt{a}') \Leftrightarrow \begin{cases} \exists\, \mathtt{l}_{\mathtt{m}_2} = (\mathtt{m},\mathtt{m}',\alpha_{\mathtt{m}_2}) \in \mathcal{L}_{\mathtt{m}} & \textbf{(iv)} \\ \exists\, \mathtt{l}_{\mathtt{e}_2} = (\mathtt{e},\mathtt{e}',\alpha_{\mathtt{e}_2},\mathtt{l}_{\mathtt{m}_2}) \in \mathcal{L}_{\mathtt{e}} & \textbf{(v)} \\ \exists\, \mathtt{l}_{\mathtt{a}_2} = (\mathtt{A}_{\mathtt{s}},\mathtt{a}',\alpha_{\mathtt{a}_2},\mathtt{l}_{\mathtt{e}_2},\mathtt{f}_2) \in \mathcal{L}_{\mathtt{a}} & \textbf{(vi)} \end{cases}$$

Or **(i)** et **(iv)** entraînent $\mathtt{l}_{\mathtt{m}_1} = \mathtt{l}_{\mathtt{m}_2} = \mathtt{l}_{\mathtt{m}}$ (i.e $\alpha_{\mathtt{m}_1} = \alpha_{\mathtt{m}_2}$) **(vii)**.
En effet, $\alpha_{\mathtt{m}_1} \neq \alpha_{\mathtt{m}_2}$ violerait la règle d'unicité du couple $(\mathtt{m},\mathtt{m}')$ lors de la création d'un *mLink* (Cf. section 4.4.2.1) et signifierait ainsi que les appariements $\mathtt{l}_{\mathtt{m}_1}$ et $\mathtt{l}_{\mathtt{m}_2}$sont contradictoires.

De façon analogue, **(ii)**, **(v)** et **(vii)** entraînent $\mathtt{l}_{\mathtt{e}_1} = \mathtt{l}_{\mathtt{e}_2} = \mathtt{l}_{\mathtt{e}}$ (i.e $\alpha_{\mathtt{e}_1} = \alpha_{\mathtt{e}_2}$) **(viii)**.

En effet, $\alpha_{\mathtt{e}_1} \neq \alpha_{\mathtt{e}_2}$ violerait la règle d'unicité du triplet $(\mathtt{e},\mathtt{e}',\mathtt{l}_{\mathtt{m}})$ lors de la création d'un *eLink* (Cf. section 4.4.2.1) et signifierait ainsi que les appariements $\mathtt{l}_{\mathtt{e}_1}$ et $\mathtt{l}_{\mathtt{e}_2}$ sont contradictoires.

Enfin, **(iii)**, **(vi)** et **(viii)** entraînent $\mathtt{l}_{\mathtt{a}_1} = \mathtt{l}_{\mathtt{a}_2}$ (i.e $\alpha_{\mathtt{a}_1} = \alpha_{\mathtt{a}_2}$, $\mathtt{A}_{\mathtt{s}_1} = \mathtt{A}_{\mathtt{s}_2}$ et $\mathtt{f}_1 = \mathtt{f}_2$).

En effet, $\mathtt{f}_1 \neq \mathtt{f}_2$ (respectivement $\alpha_{\mathtt{a}_1} \neq \alpha_{\mathtt{a}_2}$ ou $\mathtt{A}_{\mathtt{s}_1} \neq \mathtt{A}_{\mathtt{s}_2}$) violerait la règle d'unicité du couple $(\mathtt{a}',\mathtt{l}_{\mathtt{e}})$ lors de la création d'un *aLink* (Cf. section 4.4.2.1) et signifierait ainsi que les appariements $\mathtt{l}_{\mathtt{a}_1}$ et $\mathtt{l}_{\mathtt{a}_2}$ sont contradictoires.

Comme les appariements contradictoires sont interdits, on a $\mathtt{f}_1 = \mathtt{f}_2$. D'où $\|\mathtt{mapA}\varepsilon(\mathtt{e},\mathtt{a}')\| = 1$.

■

Remarque. Dans la suite de ce mémoire, nous supposons l'existence d'une expression de valeur `NULL`. Cette expression symbolise le fait qu'il n'existe pas d'appariement entre un ensemble d'attributs sources (d'une entité $\mathtt{e}$) avec un attribut cible $\mathtt{a}'$. Dans ce cas, $\mathtt{mapA}\varepsilon(\mathtt{e},\mathtt{a}') = \{\mathtt{NULL}\}$ et le théorème 5.4.2.1 se généralise au cas de figure où il n'existe pas appariement.

5.4.2.2 Opérations de MapAlgebra

MapAlgebra est constituée d'opérations qui prennent en compte la notion de mappings. Une relation de l'algèbre relationnelle est comparable à une entité de *MapAlgebra* à la différence que, lors de l'utilisation d'une entité, *MapAlgebra* exploite également les mappings avec des entités se trouvant dans d'autres modèles. Nous utilisons d'ailleurs le terme *interrogation virtuelle* d'un ensemble de modèles $\{\mathtt{m}_1, \mathtt{m}_2, ..., \mathtt{m}_n\}$ à partir d'un modèle `m` pour désigner la propagation d'une requête appliquée au modèle `m` sur chacun des modèles $\mathtt{m}_i \in \{\mathtt{m}_1, \mathtt{m}_2, ..., \mathtt{m}_n\}$ accessibles à partir du modèle `m` en utilisant les graphes d'appariements.

Pour expliciter la différence entre les opérations de l'algèbre relationnelle et celles de *MapAlgebra*, nous avons adopté la convention de nommage qui consiste à préfixer chaque opération de l'algèbre relationnelle par le mot `"Map"`. *MapAlgebra* s'inspire également de l'algèbre multi-relationnelle proposée par [Grant et al., 1993] pour les multibases de données. Par convention, dans l'algèbre multi-relationnelle, les relations et les attributs de même sens possèdent le même nom dans toutes les bases de données constituant la multibase. Cette convention est une façon implicite de représenter les mappings entre les schémas des différentes bases de données. Dans *MapAlgebra*, les mappings sont représentés explicitement et sont manipulables comme des objets, nous permettant, par exemple, d'utiliser des expressions pour exprimer les appariements d'attributs. Le but de cette section est donc d'expliciter la réécriture algébrique des opérateurs de *MapAlgebra* en fonction des opérateurs de l'algèbre relationnelle dont la traduction en requête SQL est triviale.

1. **Map-Sélection.**
   A la différence de l'algèbre relationnelle, une *Map-Sélection* des tuples d'une entité cible `e` selon le prédicat `Pred` et *"virtuellement"* sur un ensemble $\{\mathtt{m}_1, \mathtt{m}_2, ..., \mathtt{m}_n\}$ de modèles sources propage également la sélection sur l'ensemble $\{\mathtt{m}_1, \mathtt{m}_2, ..., \mathtt{m}_n\}$ de modèles, conformément aux mappings existants entre le modèle de l'entité `e` et chacun des modèles sources $\mathtt{m}_i$.

   Dans une *Map-Sélection*, après avoir traduit l'entité interrogée en entités qui lui sont appariées dans les différents modèles, il est également nécessaire, de traduire le prédicat de sélection puisque ce prédicat porte sur des attributs dont il faut déterminer les attributs appariés dans chacun des modèles sources. Le prédicat obtenu par traduction est dit *bien défini* si et seulement si chaque attribut du prédicat est apparié à un attribut dans le modèle de propagation. En d'autres termes, le prédicat résultant de la traduction ne doit pas comporter de valeurs `null` (exprimant le fait qu'un attribut du prédicat initial n'a pas pu être traduit). Dans le cas contraire, c'est-à-dire, s'il existe au moins une traduction retournant `null`, on dit que le prédicat résultant de cette propagation est *mal défini*. *MapAlgebra* enrichit donc la définition d'un prédicat (Cf. définition 5.4.1) en y ajoutant les notions de *"bien et mal défini"*.

   De plus, nous avons besoin d'une application $\mathtt{translatePred} : \mathcal{P} \times \mathcal{E} \rightarrow \mathcal{P}$ qui utilise le constructeur $\mathtt{mapA}\varepsilon$ pour traduire un prédicat sur une entité. En effet, soit `p` un prédicat et `e` une entité, `translatePred(p,e)` s'obtient en remplaçant chaque attribut `a` du prédicat `p` par l'unique élément de $\mathtt{mapA}\varepsilon(\mathtt{e}, \mathtt{a})$ (unicité garantie par le théorème 5.4.2.1).

   Nous pouvons donc formellement représenter une *Map-Sélection* par :

$$\texttt{MapSELECT}(\texttt{e}, \texttt{Pred}, \{\texttt{m}_1, \texttt{m}_2, ..., \texttt{m}_n\}) = \{\texttt{t} \in \texttt{e} \mid \texttt{Pred}(\texttt{t}) = \texttt{true}\} \cup$$

$$\left\{ \texttt{t}_1 \in \bigcup_{\texttt{e}_j \in \texttt{mapE}(\texttt{m}_1,\texttt{e})} \texttt{e}_j \mid \texttt{Pred}_1(\texttt{t}_1) = \texttt{true avec Pred}_1 = \texttt{translatePred}(\texttt{Pred}, \texttt{e}_j) \right\} \cup$$

$$\left\{ \texttt{t}_2 \in \bigcup_{\texttt{e}_j \in \texttt{mapE}(\texttt{m}_2,\texttt{e})} \texttt{e}_j \mid \texttt{Pred}_2(\texttt{t}_2) = \texttt{true avec Pred}_2 = \texttt{translatePred}(\texttt{Pred}, \texttt{e}_j) \right\} \cup$$

...

$$\left\{ \texttt{t}_n \in \bigcup_{\texttt{e}_j \in \texttt{mapE}(\texttt{m}_n,\texttt{e})} \texttt{e}_j \mid \texttt{Pred}_n(\texttt{t}_n) = \texttt{true avec Pred}_n = \texttt{translatePred}(\texttt{Pred}, \texttt{e}_j) \right\}$$

$$= \texttt{SELECT}(\texttt{e}, \texttt{Pred}) \cup \left(\bigcup_{\texttt{i}=1}^{\texttt{n}} \bigcup_{\texttt{e}_j \in \texttt{mapE}(\texttt{m}_i,\texttt{e})} \texttt{SELECT}(\texttt{e}_j, \texttt{Pred}_j) \right) \text{ avec :}$$

$$\texttt{Pred}_j = \texttt{translatePred}(\texttt{Pred}, \texttt{e}_j)$$

Le tableau 5.18 illustre l'utilisation de l'opérateur MapSELECT, dans le cas de la plate-forme DaFOE, où l'on dispose de trois modèles (le modèle terminologique (TM), le modèle termino-ontologique (TOM) et le modèle ontologique (OM).

Tableau 5.18 – Illustrattion de l'opérateur MapSELECT.

```
MapSELECT(Class,relance ≥ 0.1,{TOM,TM})= SELECT(Class,relance ≥ 0.1) ∪
                  SELECT(TerminoConcept,rate/100 ≥ 0.1) ∪
                  SELECT(Term,(frequency+variant_number)/100 ≥ 0.1 ) ∪ ...
```

Remarque. A partir de l'opérateur MapSELECT, on définit l'opérateur MapSELECT* (appelé *version étoilée* de MapSELECT) correspondant à la fermeture transitive de MapSELECT. Cet opérateur permet de propager transitivement l'appariement de l'entité e sur la totalité du graphe d'appariements. MapSELECT* permet de formaliser la clause DEPTH du langage MQL en bornant le calcul de la fermeture transitive.

2. **Map-Projection.**
A la différence de l'algèbre relationnelle, une *Map-Projection* d'une entité cible e suivant un ensemble A d'attributs et *"virtuellement"* sur un ensemble $\{\texttt{m}_1, \texttt{m}_2, ..., \texttt{m}_n\}$ de modèles sources, propage également la projection sur l'ensemble $\{\texttt{m}_1, \texttt{m}_2, ..., \texttt{m}_n\}$ de modèles, conformément aux mappings existants entre le modèle de l'entité e et chacun des modèles $\texttt{m}_i$. De manière formelle, une *Map-Projection* est définie par :

$$\texttt{MapPROJECT}(\texttt{e}, \texttt{A}, \{\texttt{m}_1, \texttt{m}_2, ..., \texttt{m}_n\}) = \{\langle \texttt{t.A} \rangle \mid \texttt{t} \in \texttt{e}\} \cup$$

$$\left\{\langle \mathtt{t.A_1}\rangle \mid \mathtt{A_1} = \left\{\mathtt{a_{1j}} \in \bigcup_{\mathtt{e_j}\in\, \mathtt{mapE(m_1,e)}} \{\mathtt{mapA(e_j,a)}\}\ \mathtt{avec\ a} \in \mathtt{A}\ \wedge\ \mathtt{t} \in \mathtt{e_j}\right\}\right\} \cup$$

$$\left\{\langle \mathtt{t.A_2}\rangle \mid \mathtt{A_2} = \left\{\mathtt{a_{2j}} \in \bigcup_{\mathtt{e_j}\in\, \mathtt{mapE(m_2,e)}} \{\mathtt{mapA(e_j,a)}\}\ \mathtt{avec\ a} \in \mathtt{A}\ \wedge\ \mathtt{t} \in \mathtt{e_j}\right\}\right\} \cup$$

...

$$\left\{\langle \mathtt{t.A_n}\rangle \mid \mathtt{A_n} = \left\{\mathtt{a_{nj}} \in \bigcup_{\mathtt{e_j}\in\, \mathtt{mapE(m_n,e)}} \{\mathtt{mapA(e_j,a)}\}\ \mathtt{avec\ a} \in \mathtt{A}\ \wedge\ \mathtt{t} \in \mathtt{e_j}\right\}\right\}$$

$$= \mathtt{PROJECT(e,A)} \cup \left(\bigcup_{\mathtt{i}=1}^{\mathtt{n}} \bigcup_{\mathtt{e_j}\in\, \mathtt{mapE(m_i,e)}} \mathtt{PROJECT(e_j,A_j)}\right) \text{ avec :}$$

$$\mathtt{A_j} = \left\{\mathtt{a_{ij}} \in \bigcup_{\mathtt{e_j}\in\, \mathtt{mapE(m_i,e)}} \{\mathtt{mapA(e_j,a)}\} \mid \mathtt{a} \in \mathtt{A}\right\}$$

Le tableau 5.19 illustre l'utilisation de l'opérateur `MapPROJECT`, dans le cas de la plate-forme DaFOE.

Tableau 5.19 – Illustrattion de l'opérateur `MapPROJECT`.

```
MapPROJECT(Class,{c_label,relance},{TOM,TM})= PROJECT(Class,{c_label,relevance}) ∪
                  PROJECT(TerminoConcept,{tc_label,rate}) ∪
                  PROJECT(Term,{t_label,frequency,variant_number}) ∪ ...
```

Remarque. A partir de l'opérateur `MapPROJECT`, on définit l'opérateur `MapPROJECT`* (appelé *version étoilée* de `MapPROJECT`) correspondant à la fermeture transitive de `MapPROJECT`. Cet opérateur permet de propager transitivement l'appariement de l'entité `e` sur la totalité du graphe d'appariements. `MapPROJECT`* permet de formaliser la clause `DEPTH` du langage MQL en bornant le calcul de la fermeture transitive.

3. **Map-Produit.**
A la différence de l'algèbre relationnelle, un *Map-Produit* de deux entités cibles `e` et `e'` *"virtuellement"* sur un ensemble $\{\mathtt{m_1}, \mathtt{m_2}, ..., \mathtt{m_n}\}$ de modèles sources propage également le produit cartésien sur l'ensemble $\{\mathtt{m_1}, \mathtt{m_2}, ..., \mathtt{m_n}\}$ de modèles, conformément aux mappings existants entre le modèle des entités `e` et `e'` et les modèles $\mathtt{m_i}$. De manière formelle, un *Map-Produit* est défini par :

$$\mathtt{MapPRODUCT(e,e',\{m_1,m_2,...,m_n\})} = \{(\mathtt{t_1},\mathtt{t_2}) \mid \mathtt{t_1} \in \mathtt{e}\ \wedge\ \mathtt{t_2} \in \mathtt{e'}\} \cup$$

$$\left\{(\mathtt{t_1},\mathtt{t_2}) \mid \mathtt{t_1} \in \mathtt{e_1}\ \wedge\ \mathtt{t_2} \in \mathtt{e'_1}\ \mathtt{avec}\ \mathtt{e_1} \in \mathtt{mapE(m_1,e)}\ \mathtt{et}\ \mathtt{e'_1} \in \mathtt{mapE(m_1,e')}\right\} \cup$$

$$\left\{(\mathtt{t_1},\mathtt{t_2}) \mid \mathtt{t_1} \in \mathtt{e_2}\ \wedge\ \mathtt{t_2} \in \mathtt{e'_2}\ \mathtt{avec}\ \mathtt{e_2} \in \mathtt{mapE(m_2,e)}\ \mathtt{et}\ \mathtt{e'_2} \in \mathtt{mapE(m_2,e')}\right\} \cup$$

...

$\{(t_1, t_2) \mid t_1 \in e_n \ \wedge \ t_2 \in e'_n \text{ avec } e_n \in \texttt{mapE}(m_n, e) \text{ et } e'_n \in \texttt{mapE}(m_n, e')\}$

$$= \texttt{PRODUCT}(e, e') \cup \left(\bigcup_{i=1}^{n} \bigcup_{e_j \in \texttt{mapE}(m_i, e)} \bigcup_{e'_k \in \texttt{mapE}(m_i, e')} \texttt{PRODUCT}(e_j, e'_k) \right)$$

Le tableau 5.20 illustre l'utilisation de l'opérateur MapPRODUCT, dans le cas de la plate-forme DaFOE.

Tableau 5.20 – Illustrattion de l'opérateur MapPRODUCT.

```
MapPRODUCT(Class,Property,{TOM,TM})= PRODUCT(Class,Property) ∪
                  PRODUCT(TerminoConcept,TerminoConceptRelation) ∪
                  PRODUCT(Term,TermRelation) ∪ ...
```

Remarque. A partir de l'opérateur MapPRODUCT, on définit l'opérateur MapPRODUCT* (appelé *version étoilée* de MapPRODUCT) correspondant à la fermeture transitive de MapPRODUCT. Cet opérateur permet de propager transitivement l'appariement de l'entité e sur la totalité du graphe d'appariements. MapPRODUCT* permet de formaliser la clause DEPTH du langage MQL en bornant le calcul de la fermeture transitive.

4. **Map-Intersection.**
A la différence de l'algèbre relationnelle, une *Map-Intersection* de deux entités cibles e et e' *"virtuellement"* sur un ensemble $\{m_1, m_2, ..., m_n\}$ de modèles sources propage également l'intersection sur l'ensemble $\{m_1, m_2, ..., m_n\}$ de modèles, conformément aux mappings existants entre les modèles des entités e et e' et les modèles m_i. De manière formelle, une *Map-Intersection* est définie par :

$\texttt{MapINTERSECT}(e, e', \{m_1, m_2, ..., m_n\}) = \{t \mid t \in e \ \wedge \ t \in e'\} \cup$

$\left\{t_1 \mid t_1 \in e_1 \ \wedge \ t_1 \in e'_1 \text{ avec } e_1 \in \texttt{mapE}(m_1, e) \text{ et } e'_1 \in \texttt{mapE}(m_1, e')\right\} \cup$

$\left\{t_2 \mid t_2 \in e_2 \ \wedge \ t_2 \in e'_2 \text{ avec } e_2 \in \texttt{mapE}(m_2, e) \text{ et } e'_2 \in \texttt{mapE}(m_2, e')\right\} \cup$

...

$\{t_n \mid t_n \in e_n \ \wedge \ t_n \in e'_n \text{ avec } e_n \in \texttt{mapE}(m_n, e) \text{ et } e'_n \in \texttt{mapE}(m_n, e')\}$

$$= \texttt{INTERSECT}(e, e') \cup \left(\bigcup_{i=1}^{n} \bigcup_{e_j \in \texttt{mapE}(m_i, e)} \bigcup_{e'_k \in \texttt{mapE}(m_i, e')} \texttt{INTERSECT}(e_j, e'_k) \right).$$

Le tableau 5.21 illustre l'utilisation de l'opérateur MapINTERSECT, dans le cas de la plate-forme DaFOE.

Tableau 5.21 – Illustrattion de l'opérateur MapINTERSECT.

MapINTERSECT(Class,Property,{TOM,TM})= INTERSECT(Class,Property) ∪ INTERSECT(TerminoConcept,TerminoConceptRelation) ∪ INTERSECT(Term,TermRelation) ∪ ...

Remarque. A partir de l'opérateur MapINTERSECT, on définit l'opérateur MapINTERSECT* (appelé *version étoilée* de MapINTERSECT) correspondant à la fermeture transitive de MapINTERSECT. Cet opérateur permet de propager transitivement l'appariement de l'entité e sur la totalité du graphe d'appariements. MapINTERSECT* permet de formaliser la clause DEPTH du langage MQL en bornant le calcul de la fermeture transitive.

5. **Map-Union.**
A la différence de l'algèbre relationnelle, une *Map-Union* de deux entités cibles e et e' *"virtuellement"* sur un ensemble $\{m_1, m_2, ..., m_n\}$ de modèles sources propage également l'union sur l'ensemble $\{m_1, m_2, ..., m_n\}$ de modèles conformément aux mappings existants entre les entités e et e' et les entités des modèles m_i. De manière formelle, une *Map-Union* est définie par :

$$\texttt{MapUNION}(e, e', \{m_1, m_2, ..., m_n\}) = \{t \mid t \in e \ \vee \ t \in e'\} \cup$$

$$\left\{t_1 \mid t_1 \in e_1 \ \vee \ t_1 \in e'_1 \text{ avec } e_1 \in \texttt{mapE}(m_1, e) \text{ et } e'_1 \in \texttt{mapE}(m_1, e')\right\} \cup$$

$$\left\{t_2 \mid t_2 \in e_2 \ \vee \ t_2 \in e'_2 \text{ avec } e_2 \in \texttt{mapE}(m_2, e) \text{ et } e'_2 \in \texttt{mapE}(m_2, e')\right\} \cup$$

...

$$\{t_n \mid t_n \in e_n \ \vee \ t_n \in e'_n \text{ avec } e_n \in \texttt{mapE}(m_n, e) \text{ et } e'_n \in \texttt{mapE}(m_n, e)\}$$

$$= \texttt{UNION}(e, e') \cup \left(\bigcup_{i=1}^{n} \bigcup_{e_j \in \texttt{mapE}(m_i, e)} \bigcup_{e'_k \in \texttt{mapE}(m_i, e')} \texttt{UNION}(e_j, e'_k) \right).$$

Le tableau 5.22 illustre l'utilisation de l'opérateur MapUNION, dans le cas de la plate-forme DaFOE.

Tableau 5.22 – Illustrattion de l'opérateur MapUNION.

MapUNION(Class,Property,{TOM,TM})= UNION(Class,Property) ∪ UNION(TerminoConcept,TerminoConceptRelation) ∪ UNION(Term,TermRelation) ∪ ...

Remarque. A partir de l'opérateur `MapUNION`, on définit l'opérateur `MapUNION`* (appelé *version étoilée* de `MapUNION`) correspondant à la fermeture transitive de `MapUNION`. Cet opérateur permet de propager transitivement l'appariement de l'entité `e` sur la totalité du graphe d'appariements. `MapUNION`* permet de formaliser la clause `DEPTH` du langage MQL en bornant le calcul de la fermeture transitive.

Proposition 5.4.2.1

Toute requête MQL se réécrit en une ou plusieurs requête(s) SQL. En effet, nous avons montré que chacune des opérations de MapAlgebra est décomposable en réunions d'opérations de l'algèbre relationnelle.

Par simplification d'écriture, nous n'avons pas, dans la présentation des opérations de *MapAlgebra*, inclu toutes les clauses que nous avons proposées pour la prise en compte des mappings dans l'interrogation des données. Le section suivante, dans laquelle nous abordons les questions d'interprétation et d'évaluation des requêtes MQL, explique davantage l'utilisation de ces clauses.

5.5 Interprétation et évaluation des requêtes MQL

Dans la section 5.4, nous avons présenté la sémantique algébrique (l'algèbre *MapAlgebra*) sur laquelle s'appuie le langage MQL. Comme nous le verrons dans le chapitre suivant (Cf. section 6.3) l'intérêt d'une telle sémantique est l'approche générique qui en découle et qui facilite l'adaptation de notre proposition en fonction des environnements de persistance. Dans cette section, nous présentons quelques définitions et propriétés nécessaires à l'interprétation et à l'évaluation des requêtes MQL.

Définition 5.5.1

De manière formelle, la forme syntaxique d'une requête MQL d'interrogation des données sur un modèle `m` *est définie par* $\texttt{mQ} = (\mathcal{A}_Q, \mathcal{E}_Q, \texttt{Pred}, \mathcal{M}_Q, \alpha, \mathcal{E}_\mathcal{F}, \texttt{depth}, \texttt{mPred})$, *où :*

- $\mathcal{A}_Q$ *représente l'ensemble des attributs de projection de la clause* `SELECT` *;*
- $\mathcal{E}_Q$ *représente les entités cibles interrogées ;*
- `Pred` *représente l'ensemble des prédicats, c'est-à-dire les triplets (attribut, comparateur, valeur), interprété comme conjonction de prédicats, et où*
  comparateur $\in \{=, \leq, \geq, \prec, \succ, ...\}$ *;*
- $\mathcal{M}_Q$ *représente l'ensemble des modèles sources sur lesquels la requête MQL doit être propagée ;*
- α *est le seuil admissible pour le degré de confiance. En effet, lors d'une propagation de la requête MQL par transitivité, le seuil* α *sera utilisé pour valider les propagations ;*

– $\mathcal{E}_{\mathcal{F}}$ *permet de spécifier un sous-ensemble d'entités sur lesquelles la requête MQL doit être propagée.*

– $\texttt{depth} \in \mathbb{N}$ *spécifie la profondeur admissible par transitivité ;*

– $\texttt{mPred}$ *représente l'ensemble des prédicats applicables sur la partie correspondant à l'extension du méta-modèle noyau.*

Remarque. Soit $\texttt{q}$ une requête MQL. Pour accéder aux paramètres de la requête $\texttt{q}$, nous utilisons la notation $\texttt{q[nom\_du\_paramètre]}$. Par exemple, $\texttt{q}[\mathcal{A}_{\texttt{Q}}]$ permet d'accéder aux attributs de la clause `SELECT` de la requête MQL.

Lors du parcours des graphes d'appariements en vue de propager une requête MQL sur d'autres modèles, des contrôles sont effectués pour garantir que les clauses `MATCH, FILTER, DEPTH,` etc. ne sont pas violées. Nous avons identifié plusieurs types de contrôles, que nous définissons pour une requête MQL écrite sur un modèle cible et devant être propagée sur des modèles sources.

Définition 5.5.2

Contrôle orienté modèle

Soit $\texttt{q}$ *une requête MQL permettant d'interroger un modèle cible* $\texttt{m'}$.

❶ *Un modèle* $\texttt{m}$*, spécifié dans la clause* `MATCH` *(*$\texttt{q}[\mathcal{M}_Q]$*), est valide par rapport à cette clause (match-valid) s'il existe un chemin (c'est-à-dire un appariement de modèles) de* $\texttt{m}$ *à* $\texttt{m'}$ *;*

❷ *Un modèle* $\texttt{m}$*, spécifié dans la clause* `MATCH` *(*$\texttt{q}[\mathcal{M}_Q]$*), est valide par rapport à la clause* `CONFIDENCE` *(*$\alpha-$*valid) s'il existe un chemin de* $\texttt{m}$ *à* $\texttt{m'}$ *dont le degré de confiance est supérieur ou égal à* $\texttt{q}[\alpha]$ *i.e* $\exists \texttt{l} = (\texttt{m}, \texttt{m}', \alpha_{\texttt{m}}) \in \mathcal{L}_{\texttt{m}}$ *tel que* $\alpha_{\texttt{m}} \geq \texttt{q}[\alpha]$ *;*

❸ *Un modèle* $\texttt{m}$*, spécifié dans la clause* `MATCH` *(*$\texttt{q}[\mathcal{M}_Q]$*), est valide par rapport à la clause* `DEPTH` *(depth-valid) s'il existe un chemin de* $\texttt{m}$ *à* $\texttt{m'}$ *dont la longueur est inférieure ou égale à la valeur spécifiée dans la clause* `DEPTH` *(*$\texttt{q[depth]}$*) de la requête* $\texttt{q}$ *i.e* $\exists \texttt{l}_{\texttt{m}}, \texttt{l}_{\texttt{m}_1}, ..., \texttt{l}_{\texttt{m}_n} \in \mathcal{L}_{\texttt{m}}$ *tel que* $\texttt{l}_{\texttt{m}} = \texttt{l}_{\texttt{m}_1} \circ ... \circ \texttt{l}_{\texttt{m}_n} = (\texttt{m}, \texttt{m}', \alpha)$ *et* $1 \leq \texttt{n} \leq \texttt{q[depth]}$ *;*

❹ *Un modèle* $\texttt{m}$*, spécifié dans la clause* `MATCH` *(*$\texttt{q}[\mathcal{M}_Q]$*), est valide par rapport à la clause* `mWHERE` *(mwhere-valid) est s'il existe un chemin de* $\texttt{m}$ *à* $\texttt{m'}$ *tel que le prédicat spécifié dans la clause* `mWHERE` *(*$\texttt{q[mPred]}$*) est satisfait ;*

❺ *Un modèle spécifié dans la clause* `MATCH` *(*$\texttt{q}[\mathcal{M}_Q]$*) est valide si ce modèle est à la fois match-valid,* α *− valid, depth-valid et mwhere-valid.*

Définition 5.5.3

Contrôle orienté entité

Soient $\texttt{q}$ *une requête MQL permettant d'interroger une entité* $\texttt{e'}$ *d'un modèle* $\texttt{m'}$.

❶ *Une entité* $\texttt{e}$ *(d'un modèle* $\texttt{m'}$*), spécifiée dans la clause* `FILTER` *(*$\texttt{q}[\mathcal{E}_Q]$*), est valide par rapport à cette clause (filter-valid) si le modèle* $\texttt{m}$ *est valide dans la requête* $\texttt{q}$ *et s'il existe un chemin*

(c'est-à-dire un appariement d'entités) de e *à* e' *;*

❷ *Une entité* e *(d'un modèle* m' *), spécifiée dans la clause* FILTER *(*q[$\mathcal{E}_Q$]*), est valide par rapport à la clause* CONFIDENCE *(α − valid) si le modèle* m *est valide dans la requête* q *et s'il existe un chemin de* e *à* e' *dont le degré de confiance est supérieur ou égal à* q[α] *i.e* $\exists \mathtt{l}_\mathtt{m} = (\mathtt{m}, \mathtt{m}', \alpha_\mathtt{m}) \in \mathcal{L}_\mathtt{m}$ *tel que* $\alpha_\mathtt{m} \geq \mathtt{q}[\alpha]$ *et* $\exists \mathtt{l}_\mathtt{e} = (\mathtt{e}, \mathtt{e}', \alpha_\mathtt{e}, \mathtt{l}_\mathtt{m}) \in \mathcal{L}_\mathtt{e}$ *tel que* $\alpha_\mathtt{e} \geq \mathtt{q}[\alpha]$ *;*

❸ *Une entité* e *(d'un modèle* m' *), spécifiée dans la clause* FILTER *(*q[$\mathcal{E}_Q$]*), est valide par rapport à la clause* DEPTH *(depth-valid) si le modèle* m *est valide dans la requête* q *et s'il existe un chemin de* e *à* e' *dont la longueur est inférieure ou égale à la valeur spécifiée dans la clause* DEPTH *(*q[depth]*) de la requête* q *i.e*

$$\begin{cases} \bullet\ \exists \mathtt{l}_\mathtt{m} = (\mathtt{m}, \mathtt{m}', \alpha_\mathtt{m}) \in \mathcal{L}_\mathtt{m} \\ \bullet\ \exists \mathtt{l}_\mathtt{e}, \mathtt{l}_{\mathtt{e}_1}, ..., \mathtt{l}_{\mathtt{e}_n} \in \mathcal{L}_\mathtt{e} \text{ tel que } \mathtt{l}_\mathtt{e} = \mathtt{l}_{\mathtt{e}_1} \circ ... \circ \mathtt{l}_{\mathtt{e}_n} = (\mathtt{e}, \mathtt{e}', \alpha, \mathtt{l}_\mathtt{m}) \text{ et } 1 \leq \mathtt{n} \leq \mathtt{q}[\mathtt{depth}] \end{cases}$$

❹ *Une entité spécifiée dans la clause* FILTER *(*q[$\mathcal{E}_Q$]*) est valide par rapport à une requête si cette entité est à la fois filter-valid, α − valid et depth-valid.*

Définition 5.5.4

Contrôle orienté attribut

Soit q *une requête MQL permettant d'extraire un attribut* a' *d'une entité* e'= dom(a')*.*

❶ *Un attribut* a *(d'une entité* e= dom(a)*) est select-valid par rapport à l'attribut* a' *spécifiée dans la clause* SELECT *(*q[$\mathcal{A}_Q$]*) l'entité* e *est valide dans la requête* q *et s'il existe un chemin (c'est-à-dire un appariement d'attributs) d'un ensemble* $\mathtt{A}_\mathtt{s}$ *(contenant* a*) à* a' *;*

❷ *Un attribut* a *(d'une entité* e*) est α − valid par rapport à l'attribut* a' *spécifié dans la clause* SELECT *si cet attribut est select-valid grâce à un chemin dont le degré de confiance est supérieur ou égal à la valeur spécifiée dans la clause* CONFIDENCE *(*q[α]*) i.e*

$$\begin{cases} \bullet\ \exists \mathtt{l}_\mathtt{m} = (\mathtt{its\_model(e)}, \mathtt{its\_model(e')}, \alpha_\mathtt{m}) \in \mathcal{L}_\mathtt{m} \\ \bullet\ \exists \mathtt{l}_\mathtt{e} = (\mathtt{e}, \mathtt{e}', \alpha_\mathtt{e}, \mathtt{l}_\mathtt{m}) \in \mathcal{L}_\mathtt{e} \\ \bullet\ \exists \mathtt{l}_\mathtt{a} = (\mathtt{A_s}, \mathtt{a}', \alpha_\mathtt{a}, \varphi_\mathtt{a}, \mathtt{l}_\mathtt{e}) \in \mathcal{L}_\mathtt{a} \text{ tel que } \mathtt{a} \in \mathtt{A_s} \text{ et } \alpha_\mathtt{a} \geq \mathtt{q}[\alpha] \end{cases}$$

❸ *Un attribut* a *(d'une entité* e*) est depth-valid par rapport à l'attribut* a' *spécifié dans la clause* SELECT *si cet attribut est select-valid grâce à un chemin dont la longueur est inférieure ou égale à la valeur spécifiée dans la clause* DEPTH *(*q[depth]*) de la requête.*

❹ *Un attribut est valide par rapport à une requête si cet attribut est à la fois α − valid et depth-valid.*

La propagation, sur un modèle source, d'une requête MQL portant sur un modèle cible, a pour but la sélection des instances des entités appariées à l'entité ou les entités interrogée(s) dans la requête. Ces instances, que nous appelons *instances à base de mappings*, sont définies formellement de la manière suivante.

Définition 5.5.5

Instance à base de mappings

Soit $\mathtt{l}_{\mathtt{m}_{12}} = (\mathtt{m}_1, \mathtt{m}_2, \alpha_{12})$*, un mLink entre deux modèles* $\mathtt{m}_1$ *et* $\mathtt{m}_2$*. Soient* $\mathtt{e}_1 \in \mathtt{m}_1[\mathcal{E}]$ *et* $\mathtt{e}_2 \in \mathtt{m}_2[\mathcal{E}]$ *deux entités. Une instance* `i` *de* $\mathtt{e}_1$ *est une instance à base de mappings de* $\mathtt{e}_2$ *si* `i` *n'est pas une instance locale de* $\mathtt{e}_2$ *(i.e* $\mathtt{i} \notin \mathtt{e}_2$*) et s'il existe* $\mathtt{l}_\mathtt{e} = (\mathtt{e}_1, \mathtt{e}_2, \alpha, \mathtt{l}_{\mathtt{m}_{12}}) \in \mathcal{L}_\mathtt{e}$.

Pour déterminer les *instances à base de mappings* à partir d'une requête MQL, nous extrayons, puis exploitons le graphe de mappings pour propager cette requête MQL sur des modèles sources. Pour chaque propagation obtenue, nous générons la ou les requête(s) SQL à exécuter sur le modèle, et construisons également une nouvelle requête MQL que nous propageons la nouvelle requête MQL de proche en proche, en effectuant les contrôles orientés modèle, entité et attributs). L'algorithme 1 illustre le processus de génération des requêtes pour le calcul des *instances à base de mappings*. Une requête SQL est représentée par un ensemble d'attributs de la clause `SELECT`, une entité de la clause `FROM` et un ensemble de prédicats pour la clause `WHERE`. Ainsi `(Atts, Ents, Pred)` symbolise la construction d'une requête SQL où les paramètres `Atts`, `Ents` et `Pred` correspondent respectivement à l'ensemble des attributs de la clause `SELECT`, l'ensemble des entités de la clause `FROM` et l'ensemble des prédicats de sélection de la clause `WHERE`. La notation `(attribut, comparateur, valeur)` permet de construire un prédicat. Par exemple, `(attribut, =, valeur)` est interprété par "`WHERE attribut = valeur`".

Notons que, dans l'algorithme 1, les appariements *directs* sont privilégiés aux appariements *indirects* (obtenus en composant des appariements). En guise d'illustration, considérons l'exemple de la figure 5.2 (a) et supposons que l'on souhaite, pour une requête appliquée sur le modèle $\mathtt{m}_3$, propager la requête sur les modèles $\mathtt{m}_2$ et $\mathtt{m}_1$ appariés à $\mathtt{m}_3$. Si la propagation sur le modèle $\mathtt{m}_2$ peut se faire sans ambigüité (du fait de l'unicité du *mLink* $\mathtt{mL}_{23}$ existant entre les modèles $\mathtt{m}_2$ et $\mathtt{m}_3$), la propagation, sur le modèle $\mathtt{m}_1$ pose, par contre, un problème de cohérence du résultat. En effet, le modèle $\mathtt{m}_1$ est accessible de deux manières à partir du modèle $\mathtt{m}_3$:

❶ en considérerant le *mLink* direct $\mathtt{mL}_{13}$ existant entre les modèles $\mathtt{m}_1$ et $\mathtt{m}_3$;

❷ en calculant un *mLink* indirect $\mathtt{mL}' = \mathtt{mL}_{13} \circ \mathtt{mL}_{23}$.

Puisque l'opération de composition engendre un coût de calcul supplémentaire, nous avons choisi (Cf. figure 5.2 (b)), pour l'implémentation de l'algorithme 1, d'utiliser la première manière dans laquelle la terminaison de l'algorithme est garantie par un parcours du graphe avec marquage des nœuds visités. Cette approche est donc appliquée de proche en proche sur l'ensemble des nœuds du graphe de mappings jusqu'à ce que les différents noœuds accessibles à partir du modèle cible aient tous été visités.

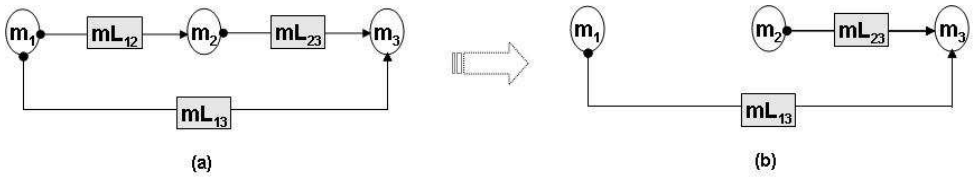

FIGURE 5.2 – Appariements directs privilégiés aux appariements indirects

Algorithme 1 MQL2SQL : Génération de requêtes SQL à partir d'une requête MQL

Input: $\mathcal{G}_m = (\mathcal{N}_m, \mathcal{L}_m)$ /* Le graphe des *mLinks* */
$\mathcal{G}_e = (\mathcal{N}_m, \mathcal{L}_e)$ /* Le graphe des *eLinks* */
$\mathcal{G}_a = (\mathcal{N}_a, \mathcal{L}_a)$ /* Le graphe des *aLinks* */
mQ_0 /* La requête MQL à propager sur des modèles sources */

Output: Q /* L'ensemble des requêtes SQL à exécuter après propagation */

1: $Q \leftarrow \{\}$
2: $\forall e \in mQ_0[\mathcal{E}_Q]$
3: $\forall l_m \in \mathcal{L}_m$ tel que $l_m[m_t] = its\_model(e)$
/* l_m peut aussi être un *mLink* obtenu par composition successive de *mLinks* */
4: **si** ($l_m[m_s]$ est valide par rapport à mQ_0) **alors**
5: $\forall l_e \in \mathcal{L}_e$ tel que $(l_e[mL] = l_m) \wedge (l_e[e_t] = e)$
/* l_e peut aussi être un *eLink* obtenu par composition successive de *eLinks* */
//A : Conversion de l'entité interrogée
6: $e' \leftarrow l_e[e_s]$
7: **si** (e' est valide par rapport à mQ_0) **alors**
8: $Atts' \leftarrow \{\}$ /* Ensemble d'attributs convertis */
//B : Conversion des attributs de la clause `SELECT`
9: $\forall a \in mQ_0[\mathcal{A}_Q]$
10: **si** (a est valide par rapport à mQ_0) **alors**
11: Soit $l_a \in \mathcal{L}_a$ tel que $(l_a[eL] = l_e) \wedge (l_a[a_t] = a)$
/* l_a peut aussi être un *aLink* obtenu par composition globale et successive de *aLinks* */
12: $Atts' \leftarrow Atts' \cup \{l_a[A_s]\}$
//C : Conversion des prédicats
13: $P' \leftarrow \{\}$ /* L'ensemble des prédicats convertis */
14: $\forall p \in mQ_0[\texttt{Pred}]$, soit $l'_a \in \mathcal{L}_a$ tel que $l'_a[a_t] = p[attribute]$
15: $p' \leftarrow (l'_a[\varphi], p[comparator], p[value])$
16: $P' \leftarrow P' \cup \{p'\}$
//D : Construction de la requête SQL
17: $q' \leftarrow (Atts', e', P')$
18: $Q \leftarrow Q \cup \{q'\}$
19: **retour Q**

Remarque. La ligne 15 de l'algorithme 1 n'est valide que si $l'_a[\varphi]$ est un attribut. L'ensemble $\mathcal{F}_{\texttt{sem}}$ des expressions sémantiques est de ce fait construit de manière à respecter cette contrainte. En effet, $\mathcal{F}_{\texttt{sem}}$ se limite aux expressions de base du langage SQL telles que l'addition, la soustraction, la division, etc. Or, si `a` et `b` sont deux attributs d'une entité `e`, la requête `"SELECT a+b FROM e WHERE a-b = xxx"` est valide selon la norme SQL, sous réserve de compatibilité du typage des attributs `a` et `b` avec les domaines de définition respectifs des opérations d'addition et de soustraction. Plus généralement, l'ensemble $\mathcal{F}_{\texttt{sem}}$ des expressions vérifie la propriété suivante.

Propriété 5.5.1

Soient $a_1, ..., a_n$, *des attributs d'une entité* e. $\forall \varphi, \varphi' \in \mathcal{F}_{sem}$, *la requête* "SELECT $\varphi(a_1, ..., a_n)$ FROM e WHERE $\varphi'(a_1, ..., a_n) = v$" *(où* v *est une valeur donnée) est valide sous réserve de compatibilité du typage des attributs* $a_1, ..., a_n$ *avec les domaines de définition respectifs des fonctions* φ *et* φ'.

5.6 Conclusion

Dans ce chapitre, nous avons identifié l'écriture des requêtes, l'exploration du graphe de mappings, la saturation mémoire et la cohérence des instances comme des problèmes posés pour l'exploitation des mappings dans une application modélisée en utilisant le mapping de modèles dans un environnement persistant de base de données. Pour résoudre ces problèmes, nous avons proposé le langage MQL qui facilite l'exploration du graphe de mappings dans un contexte persistant de bases de données. MQL est une extension du langage de requête SQL auquel nous avons ajouté des clauses spécifiques à la notion de mappings entre modèles. Pour cela, nous avons proposé, une description algébrique (l'algèbre *MapAlgebra*) et interprétative du processus de propagation de requêtes d'un modèle sur d'autres modèles. L'approche MQL que nous avons proposée exploite la souplesse offerte par les bases de données à base de modèles et permet non seulement de faire évoluer dynamiquement le *méta-modèle noyau* de la BDBM, mais aussi d'exploiter dynamiquement ces évolutions dans l'écriture des requêtes MQL. De plus, *MapAlgebra* propose un ensemble d'opérations nécessaire à la gestion des données de la plateforme DaFOE.

Chapitre 6

La plate-forme DaFOE

Sommaire

Résumé. Dans ce chapitre, nous illustrons de manière non exhaustive le fonctionnement de la plate-forme DaFOE dédiée à la construction d'ontologies à partir de textes dans le contexte de l'ingénierie des modèles. Nous y présentons également les solutions techniques mises en œuvre pour la réalisation de la plate-forme DaFOE en décrivant notamment l'architecture à base de *plug-ins* selon laquelle a été conçue la plate-forme afin de répondre au mieux à l'exigence d'extensibilité fonctionnelle de celle-ci. L'implémentation du langage MQL ainsi que l'outil support permettant de l'exploiter sont également présentés dans ce chapitre.

6.1 Introduction

Dans les chapitres précédents, nous avons présenté notre approche de modélisation du processus de construction d'ontologies à partir de textes. Le chapitre 3, par exemple, nous a permis de présenter le cadre méthodologique de construction d'ontologies ainsi que notre approche à base de mappings permettant de semi-automatiser la construction, en évoquant notamment la question de l'évolution des modèles. Dans les chapitres 4 et 5, nous avons respectivement abordé les aspects liés aussi bien à la persistance des modèles et des mappings qu'à l'exploitation des mappings notamment lors de l'interrogation des données. Ces chapitres n'ont pas abordé les problématiques techniques inhérentes à leur mise en œuvre au sein de la plate-forme DaFOE [Charlet et al., 2010b, Szulman et al., 2010, Charlet et al., 2010a]. Dans ce chapitre, nous présentons ces problématiques techniques en précisant les solutions que nous avons proposées. Dans la section 6.2 nous présentons l'architecture modulaire de la plate-forme DaFOE. Nous décrivons en suite, dans la section 6.3 l'implémentation d'un interpréteur de requêtes MQL. La problématique d'évolutivité fonctionnelle de la plate-forme ainsi que l'architecture à base de *plug-ins* que nous proposons sont présentées dans la section 6.4. Enfin, la section 6.5, pour sa part, propose une visite guidée de la plate-forme DaFOE.

6.2 Architecture de la plate-forme DaFOE

Développée en Java, la plate-forme DaFOE à été conçue suivant une architecture modulaire représenté par la figure 6.1. Cette architecture modulaire comporte trois principaux modules :

1. *un module présentation*, dédié à la conception des interfaces utilisateurs (IHM) permet d'interagir avec un utilisateur ;
2. *un module métier*, comportant toutes les fonctionnalités dont a besoin le *niveau présentation* (pour l'affichage des données, par exemple) ;
3. *un module persistance*, constitué d'une part, d'une API (Application Programming Interface) appelée DaFOE_framework proposant une représentation *objet* de la BDBM, et, d'autre part, d'une API correspondant au moteur d'interprétation des requêtes MQL. Ce niveau *persistance* possède également deux autres API : Hibernate [21] et JDBC (Java DataBase Connectivity). Ces API sont utilisées pour la manipulation des données de la BDBM. Le *framework* Hibernate permet, au moyen de fichiers de configuration XML, de définir des correspondances entre des classes Java (appelées POJOs) et le schéma d'une base de données relationnelle. Il offre ainsi un accès transparent aux données des bases de données *sans* pour autant écrire une seule ligne de code SQL car *Hibernate* génère dynamiquement les requêtes SQL d'accès aux données. Un autre avantage d'*Hibernate* repose sur son *cache d'objets* dans lequel sont stockées les données déjà extraites de la base de données, limitant ainsi les accès répétitifs à la base de données pour des objets déjà stockés en mémoire centrale. En cas de modification d'une donnée via un accès direct à la base de données, *Hibernate* assure la synchronisation de l'objet présent dans la mémoire centrale et correspondant à la donnée modifiée dans la base de données.

21. http://www.hibernate.org/

Comme l'API *Hibernate* fonctionne par configuration statique d'un mapping entre un modèle objet et le schéma d'une base de données relationnelle, cette API est utilisée pour établir un *mapping objet/relationnel* entre DaFOE_Framework et le schéma de la BDBM. De ce fait, *Hibernate* manipule uniquement la partie statique de notre modélisation, c'est-à-dire celle représentant le *méta-méta-modèle*, le *méta-modèle noyau* ainsi que les parties de base des modèles terminologique, termino-ontologique et ontologique. Pour manipuler la partie dynamique de notre modélisation, c'est-à-dire celle correspondant à la fois à l'évolution du *méta-modèle noyau* et des modèles (terminologique, termino-ontologique et ontologique), un accès direct à la BDBM est offert grâce à l'API JDBC. L'API *MQLEngine*, pour sa part, est utilisée pour les opérations exploitant les mappings. Elle permet, par exemple d'extraire les *instances à base de mappings*.

La technologie Eclipse Rich Client Platform (Eclipse RCP) [Jeff McAffer and Aniszczyk, 2010] a été utilisée pour la création des IHM et pour implémenter le mécanisme d'évolutivité fonctionnelle de la plate-forme DaFOE.

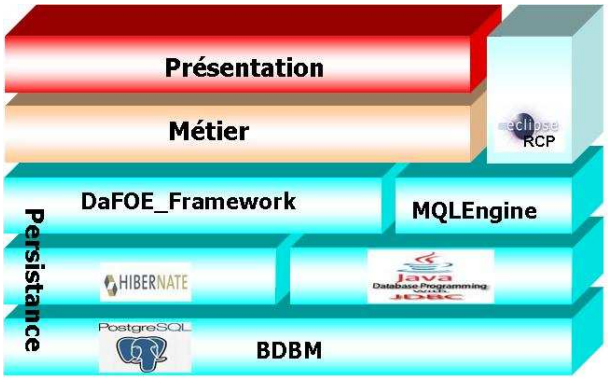

FIGURE 6.1 – Architecture multi-niveau de la plate-forme.

Pour implémenter cette architecture, nous avons essentiellement utilisé les techniques d'Ingénierie Dirigée par les Modèles (IDM). Ces techniques offrent notamment la possibilité de transformer des modèles et de générer des programmes à partir de modèles. L'environnement d'IDM que nous avons utilisé est celui d'ECCO [Staub and Maier, 1997] avec des modèles décrits dans le langage EXPRESS [IS010303.02, 1994, Schenk and Wilson, 1994]. Comme le montre la figure 6.2, le méta-méta-modèle, le méta-modèle ainsi que les modèles terminologique, termino-ontologique et ontologique sont décrit en EXPRESS. Un programme générique, écrit en EXPRESS, permet ensuite de générer des fichiers XML de mapping *hibernate*. Ces fichiers de mapping sont ensuite utilisés pour, à partir du *framework Hibernate*, générer à la fois des classes Java (permettant un accès objet à la base de données) et du code SQL (permettant, par exemple, de créer les tables de la BDBM). Enfin, le code SQL ainsi généré est exécuté sur le SGBD pour, d'une part, construire l'architecture de méta-modélisation de notre BDBM (les parties méta-schéma, schéma et instance), et, d'autre part, peupler les tables des parties méta-schéma et schéma.

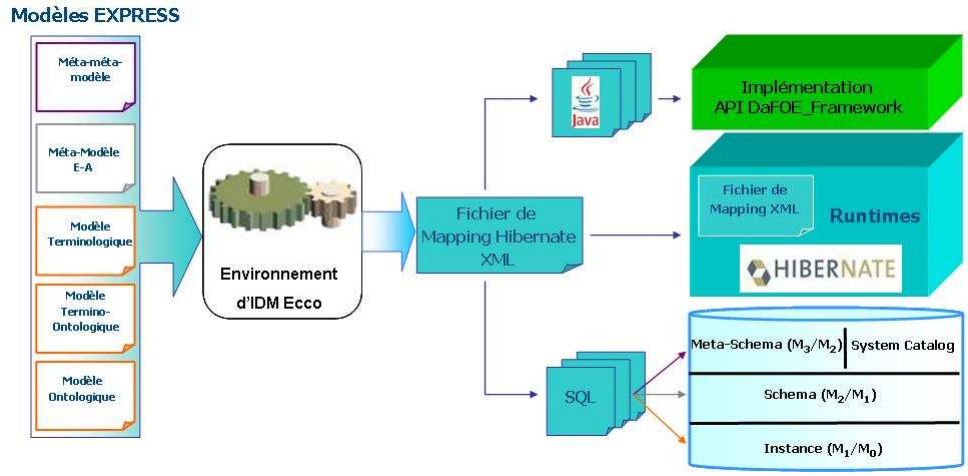

FIGURE 6.2 – IDM avec ECCO : implémentation de la plate-forme DaFOE.

6.3 Implantation d'un interpréteur de requêtes MQL

Pour la manipulation des mappings dans la plate-forme, nous avons conçu l'outil MQLToolkit qui est un plug-in pour la plate-forme DaFOE. MQLToolkit permet notamment l'édition et l'exécution des requêtes MQL ainsi que la visualisation des requêtes et des transformations. MQLToolkit comporte trois principaux modules :

❶ un module d'édition des requêtes (*MQLEditor*) et de visualisation des résultats produits par l'exécution des requêtes ;

❷ un moteur d'interprétation et d'exécution des requêtes MQL (*MQLEngine*) définissant également la grammaire du langage MQL ;

❶ un moteur d'interprétation des requêtes SQL (*SQLEngine*) permettant notamment de traduire le résultat d'une requête SQL en un résultat d'une requête MQL.

Nous avons utilisé le *framework* Xtext [Itemis, 2011] pour, d'une part, développer les IHM de *MQLEditor*, et, d'autre part, pour décrire la grammaire du langage MQL et pour réaliser l'analyse lexicale et syntaxique de ce langage.

Pour implanter le langage MQL sur notre BDBM, nous avons dû choisir une méthode pour réaliser le traitement d'une requête. On distingue deux principales approches d'implantation. La première approche consiste à traduire une requête du langage source en une requête SQL spécifique à la BDBM. Cette approche a été suivie dans les systèmes RDF-Suite [Alexaki et al., 2001] et RStar [Ma et al., 2004]. La seconde approche consiste à traduire une requête du langage source en une suite d'appels d'une API dont chaque méthode retourne un ensemble de données de la BDBM. Cette approche a été suivie dans le système Sesame [Broekstra et al., 2002].

Si la première solution présente l'avantage de profiter de l'important travail sur l'optimisation des moteurs SQL des bases de données, elle présente cependant l'inconvénient de dépendre de la BDBM sur laquelle le langage est implanté. La seconde solution quant à elle permet la portabilité sur différentes BDBM puisqu'il est possible de fournir une implémentation de l'API pour chaque BDBM. Cependant,

dans ce cas, l'optimisation des requêtes est à la charge du moteur d'interpretation du langage implanté.

Souhaitant bénéficier à la fois de l'optimisation des moteurs SQL et de la portabilité de l'implantation sur d'autres BDBM, nous avons choisi d'allier ces deux méthodes. Ainsi, le langage MQL est traduit en une requête SQL qui dépend de la BDBM. Mais, cette traduction est réalisée en encapsulant les spécificités de cette BDBM telles que :

- l'approche choisie pour représenter les données et les modèles. En effet, les BDBM proposent différentes approches de représentation pour les données et supportent différents modèles et par conséquent différentes représentations logiques pour les modèles ;
- le SGBD cible. En effet, les BDBM sont implantées sur différents SGBD qui implantent plus ou moins la norme SQL.

Pour encapsuler la représentation choisie, la traduction d'une requête MQL en SQL est réalisée en appelant des méthodes d'interfaces Java. Ces interfaces, implantées pour notre BDBM, peuvent l'être pour une autre BDBM. Pour encapsuler le SGBD cible, une requête MQL est transformée en une expression de l'algèbre *MapAlgebra* qui à son tour est transformée en une ou plusieurs expression(s) de l'algèbre relationnelle avant d'être traduite en SQL. Ainsi, un changement de SGBD ne nécessite que de remplacer la traduction d'une expression de l'algèbre relationnelle dans le langage SQL supporté par le SGBD. Cette algèbre est également réutilisable pour un autre environnement de persistance (fichier XML par exemple) à condition d'implémenter, pour ce nouvel environnement de persistance, les différents opérateurs de *MapAlgebra*.

En suivant cette approche, le processus de traitement d'une requête MQL (Cf. figure 6.3) comporte les neuf étapes suivantes :

❶ l'utilisateur saisit une requête MQL en utilisant *MQLEditor* ;

❷ la requête saisie est transmise au moteur d'interprétation *MQLEngine* ;

❸ *MQLEngine* interprète la requête et génère une opération algébrique de *MapAlgebra* correspondant à la requête ;

❹ *MQLEngine* transforme l'arbre algébrique *MapAlgebra* en un arbre algébrique utilisant des opérateurs de l'algèbre relationnelle ;

❺ un moteur d'interprétation des requêtes SQL nommé *SQLEngine* transforme l'expression de l'algèbre relationnelle en une ou plusieurs requête(s) SQL conforme(s) au SGBD PostgreSQL sur lequel notre BDBM est implanté ;

❻ la ou les requête(s) SQL est/sont ensuite exécutée(s) sur la BDBM ;

❼ le résultat retourné est transmis au moteur *MQLEngine* ;

❽ *MQLEngine* transforme le résultat en un ensemble d'informations de type `ResultSet` ;

❾ transformation de l'ensemble d'informations de type `ResultSet` pour obtenir le résultat de la requête MQL, c'est-à-dire une information de type `MQLResultSet`.

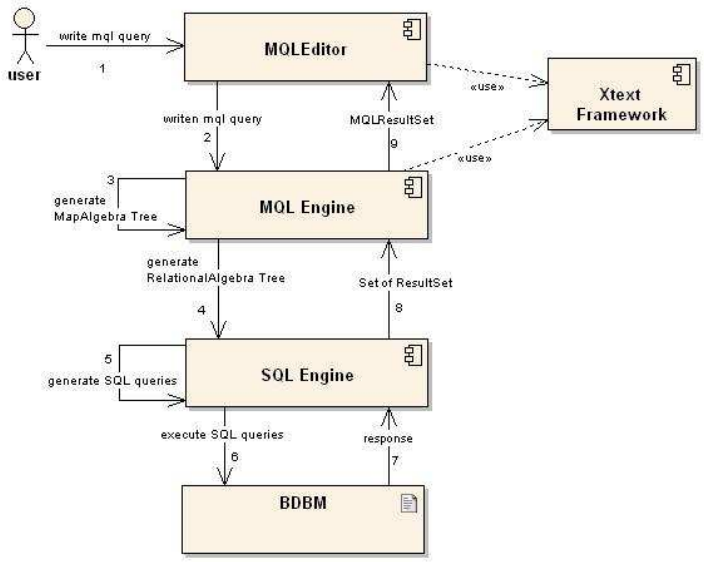

FIGURE 6.3 – Processus de traitement d'une requête MQL.

6.4 Évolutivité fonctionnelle

Pour les besoins de la plate-forme, l'une des exigences techniques était de pouvoir supporter l'ajout de nouvelles fonctionnalités à la plate-forme DaFOE. Pour cette raison, la plate-forme ne pouvait s'appuyer sur une architecture logicielle statique. Nous présentons, dans cette section, l'architecture évolutive basée sur le concept d'application extensible que nous avons proposée pour la plate-forme DaFOE.

6.4.1 Notion d'application extensible

[O'Conner, 2007] définit une application extensible comme étant une application dont les fonctionnalités peuvent facilement évoluer sans pour autant que le code de l'application de base ne soit modifié. En d'autres termes, dans ce genre d'application, l'extension ne requiert pas une recompilation de l'application de base. L'évolutivité fonctionnelle de l'application de base, réalisable par tout utilisateur *averti* (un développeur de *plug-ins*), se fait par l'ajout de nouveaux modules (ou *plug-ins*, ou extensions) sur des points d'extension prévu à cet effet. L'application de base (Cf. figure 6.4), encore appelée *application noyau*, découvre (ou détecte) les nouveaux *plug-ins* et les rend utilisables par un utilisateur. Dans la pratique, l'installation d'un nouveau module peut consister à déposer la ressource correspondant au nouveau module dans un répertoire connu de l'application de base et dédié à l'évolutivité fonctionnelle de celle-ci (un répertoire nommé "*Plug-ins*" par exemple) : c'est le mode d'évolutivité dit par scrutation. Contrairement aux applications simplement modulaires que nous appelons *"applications tout en un"*, du fait qu'il faille recompiler toute l'application lors d'un ajout de nouvelles fonctionnalités, les applications extensibles, grâce à la facilité d'installation, de mise à jour et de désinstallation des modules, facilitent à la fois l'évolutivité et la maintenance d'une application.

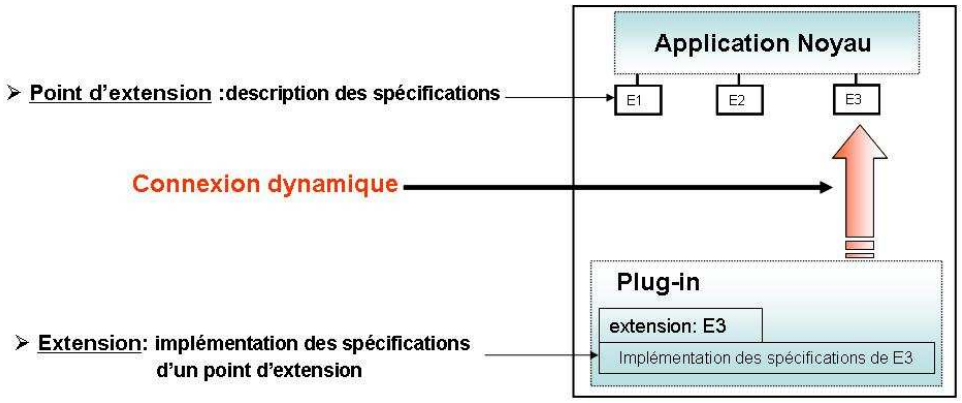

FIGURE 6.4 – Architecture d'une application extensible.

6.4.2 Proposition d'une architecture fonctionnelle flexible

Comme solution au besoin d'extensibilité fonctionnelle, nous avons proposé que la plate-forme DaFOE s'appuie sur une architecture à base de *plug-ins* (Cf. figure 6.5), centrée autour d'une *application noyau*. Inspirée de l'architecture à base de *plug-ins* de la plate-forme Eclipse [22], la plate-forme DaFOE contient, pour chaque étape du processus de construction d'ontologies, différents points d'extension. Un point d'extension est une interface représentant le "contrat" que doit remplir une extension (ou plug-in) afin d'être utilisable par *l'application noyau* comme contributeur pour un service donné. Il ne correspond pas à une fonctionnalité atomique souhaitée par l'utilisateur. Un point d'extension représente plutôt un ensemble de fonctionnalités analogues, factorisées à l'aide de critères de traitement, et caractérisées par les mêmes APIs d'accès. Pour un éditeur d'ontologies, par exemple, *l'application noyau* peut être constituée de l'ensemble des API d'accès aux classes et instances d'une ontologie. Si l'on souhaite offrir divers modes de visualisation du contenu de l'ontologie (par tableau, par graphe, etc.), un point d'extension permet, dans ce cas, de donner toutes les spécifications à remplir pour offrir ces différents modes de visualisation.

La conception d'une application extensible nécessitant de caractériser les différentes façons d'étendre l'application, nous présentons, dans la section suivante, les différents points d'extension que nous avons proposés pour la plate-forme DaFOE.

6.4.3 Points d'extension

La plate-forme DaFOE est composée, d'une part, d'une *application noyau* comportant un ensemble de fonctionnalités de base, et, d'autre part, d'une description des points d'extension permettant de rajouter des fonctionnalités complémentaires. Cette *application noyau* est également chargée de la gestion des nouvelles fonctionnalités. Pour la plate-forme DaFOE, nous avons identifié quatre catégories de points d'extension ; ceux-ci se distinguant selon qu'un *utilisateur averti* souhaite soit enrichir les interfaces graphiques de la plate-forme, soit enrichir la gestion des entrées/sorties ou encore modifier la l'environnement de persistance des données.

22. http://www.eclipse.org/

6.4.3.1 Traitements graphiques : Tabwiget_Extension_Point

Les traitements graphiques spécifient les points d'extension permettant aux extensions d'enrichir les interfaces graphiques utilisateur de la plate-forme en offrant de nouveaux modes de visualisation des données par accès au modèle de données. Le plug-in *GraphVIZ-Plug-in* (Cf. figure 6.5), par exemple, offre une visualisation de l'ontologie sous forme de graphe.

6.4.3.2 Traitements des entrées de données : Import_Extension_Point

Les traitements des entrées de données spécifient les points d'extension permettant aux *plug-ins* d'importer des données au sein de la plate-forme. Dans l'étape terminologique par exemple, pour gérer la diversité des résultats fournis par les outils de TAL, un point d'extension permet de déployer des plug-ins propres à chacun de ces outils de TAL. Dans l'étape termino-ontologique, ce point d'extension offre la possibilité de créer un plug-in permettant, par exemple, d'importer un thésaurus conforme au format SKOS. Pour l'étape ontologique, où se pose le problème de diversité des modèles d'ontologies (OWL, PLIB, RDF/S , etc.), la création d'une ontologie à partir d'une ontologie existante nécessite, en fonction du modèle d'ontologies utilisé pour créer l'ontologie à importer, à créer un plug-in d'importation conforme à ce modèle d'ontologie.

6.4.3.3 Traitements des sorties de données : Export_Extension_Point

Les traitements dits de sortie des données correspondent aux traitements duaux de ceux des entrées de données. Leurs comportements sont d'ailleurs similaires, à la seule différence que les uns agissent en entrée tandis que les autres agissent en sortie.

6.4.3.4 Traitements de persistance : DataAccess_Extension_Point

Ces traitements visent à supporter la diversité des SGBD sur lesquels pourrait être déployée la base de données. Dans sa version actuelle, la persistance des données de la plate-forme est assurée par le SGBD PostgreSQL et la migration vers un autre SGBD (Oracle, par exemple) nécessite le développement et le déploiement d'un *plug-in* d'accès aux données sur ce nouveau SGBD. Ce choix technique nous assure un meilleur découplage entre les spécifications de la plate-forme DaFOE et les implémentations réalisées.

Ayant précisé les différents points d'extension de la plate-forme DaFOE, nous présentons, dans la section suivante, la conception et la réalisation de chacun de ces points d'extension.

6.4.4 Conception et réalisation

Dans cette section, nous décrivons brièvement la réalisation de la plate-forme DaFOE. Nous décrivons tout d'abord le fonctionnement du noyau de la plate-forme DaFOE, et, nous présentons ensuite la modélisation des différents points d'extension.

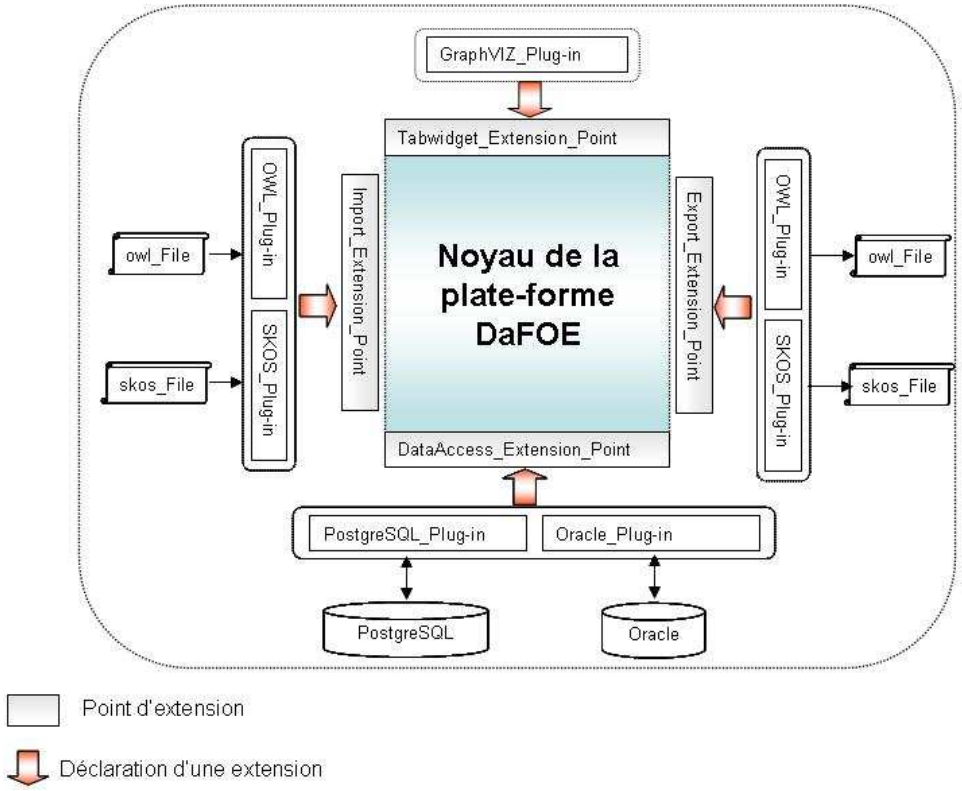

FIGURE 6.5 – Architecture à base de *plug-ins* de DaFOE.

6.4.4.1 Fonctionnement du noyau

Inspiré du fonctionnement du noyau de la plate-forme Eclipse, le noyau de la plate-forme DaFOE est chargé de :

- la détection et l'installation des *plug-ins* ;
- l'activation d'un *plug-in* lorsque celui-ci est sollicité.

En limitant le noyau de la plate-forme DaFOE au rôle d'orchestration, nous réduisons le couplage entre les différents modules de l'application, facilitant ainsi à la fois l'évolution et la maintenance de la plate-forme. Par ailleurs, un module de gestion du cycle de vie des *plug-ins*, basé sur une approche de programmation par composants et conforme à la norme OSGi [23], permet l'installation, la mise à jour et la désinstallation dynamique des *plug-ins* (ou chargement à chaud c'est-à-dire sans redémarrer la plate-forme DaFOE). Toutes ces opérations sont décrites par le diagramme d'états d'un *plug-in* (aussi appelé " *bundle*" dans la norme OSGi) de la figure 6.6. Dans ce diagramme, *Installed, Resolved, Uninstalled, etc.* représentent les états possibles d'un *plug-in*, tandis que *install, update, uninstall,* sont des actions permettant respectivement d'installer, de mettre à jour et de désinstaller un *plug-in*.

23. Open Gateway Service initiative : http://www.osgi.org/Main/HomePage

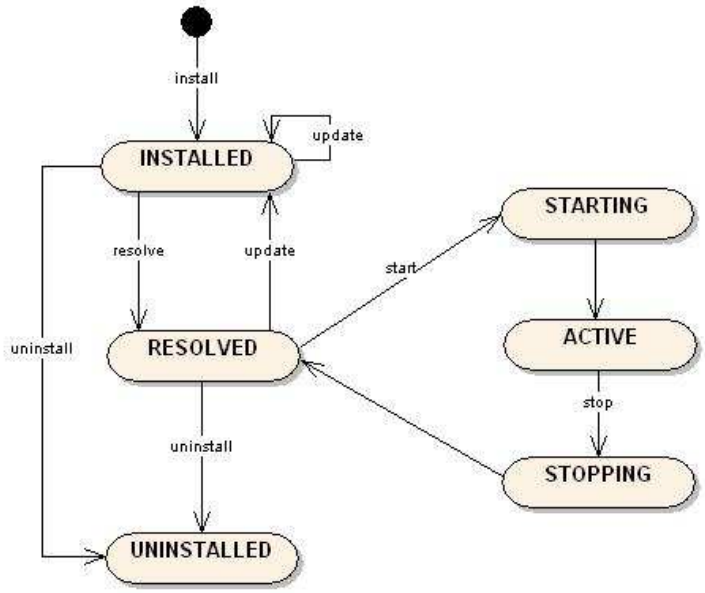

FIGURE 6.6 – Diagramme d'états d'un *plug-in*.

6.4.4.2 Points d'extension

Dans cette section, nous décrivons chacun des points d'extension que nous avons proposés. La figure figure 6.8 présente la modélisation des points d'extension et des extensions. Pour la plate-forme DaFOE, cette modélisation a été implémentée à l'aide de la technologie Eclipse RCP dans laquelle un point d'extension est décrit par un schéma XML.

1. **TabWidget_Extension_Point**
   Les *plug-ins* contributeurs à ce point d'extension (Cf. figure 6.8) doivent disposer d'une classe dite *classe point d'entrée* qui est une sous-classe de la classe abstraite *AbstractTabWidget*. Cette *classe point d'entrée* doit fournir, par exemple, une implémentation de la méthode *createUI* permettant d'initialiser l'interface graphique du *plug-in*. Le test de ce point d'extension a été réalisé en développant des *plug-ins* permettant de manipuler les données des étapes terminologique, termino-ontologique et ontologique.

2. **Import_Extension_Point**
   Comme nous l'avons vu précédemment, ce point d'extension permet d'assurer, pour la plate-forme DaFOE, une évolution fonctionnelle pour le traitement des données en entrée. Le modèle de ce point d'extension, représenté par la figure 6.8, montre qu'un *plug-in* contributeur à ce point d'extension doit disposer d'une classe dite *classe point d'entrée* et qui est sous-classe de la classe abstraite *AbstractResourceImportation*. Cette *classe point d'entrée* doit fournir, par exemple, une implémentation pour la méthode *importData* réalisant l'importation proprement dite des données.
   Pour valider ce point d'extension, nous avons réalisé :
   – un *plug-in* permettant d'importer, dans l'étape ontologique de la plate-forme DaFOE, une ontologie compatible avec un sous-ensemble fonctionnel du langage OWL. Ce *plug-in* utilise la

librairie OWL_API [24] qui est un *framework* dédié à la manipulation des ontologies conformes aux modèles d'ontologies OWL.
- un *plug-in* permettant d'importer, dans l'étape termino-ontologique de la plate-forme, un thésaurus compatible avec un sous-ensemble fonctionnel du langage SKOS. *Ce plug-in* utilise la libraire appelée Api_SKOS, que nous avons développée et qui permet de manipuler sous forme d'objets, des données représentées dans un fichier au format SKOS.
- deux plug-ins permettant d'importer, au niveau de la couche terminologique de la plate-forme, des corpus respectivement traités par les outils de TAL YaTea [Aubin and Hamon, 2006] et SYNTEX [Didier Bourigault and Ozdowska, 2005].

3. **Export_Extension_Point**
Dual du point d'extension Import_Extension_Point, son modèle est semblable à celui d'Import_-extension_Point. Comme le montre la figure 6.8, un plug-in contributeur à ce point d'extension doit disposer d'une *classe point d'entrée* qui est sous-classe de la classe abstraite *AbstractResourceExportation*. Cette *classe point d'entrée* doit fournir, par exemple, une implémentation pour la méthode *exportData* réalisant l'exportation proprement dite des données.

Pour valider ce point d'extension, nous avons réalisé :
- un *plug-in* permettant d'exporter, dans l'étape ontologique de la plate-forme DaFOE, une ontologie compatible avec un sous-ensemble fonctionnel du langage OWL ;
- un *plug-in* permettant d'exporter, dans l'étape termino-ontologique de la plate-forme, un thésaurus compatible avec un sous-ensemble fonctionnel du langage SKOS (cf. figure 6.7).

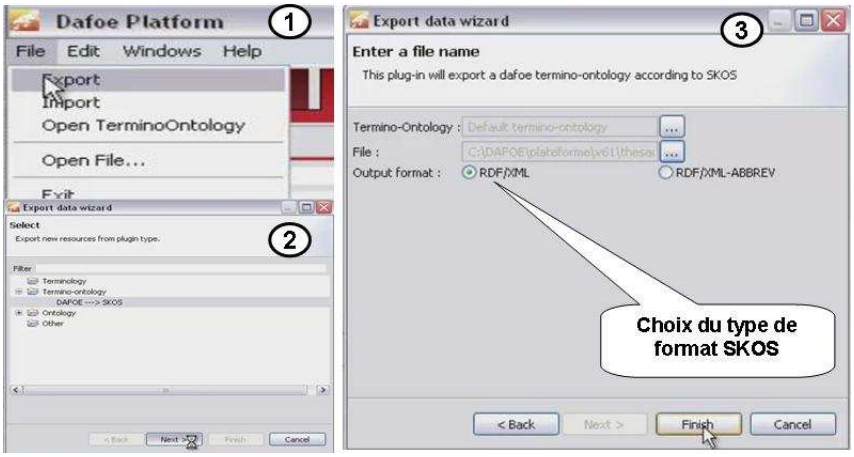

FIGURE 6.7 – Exportation d'une termino-ontologie sous la forme d'un thésaurus en SKOS

4. **DataAccess_Extension_Point**
Plus complexe à mettre en œuvre que les autres points d'extension, le développement d'un *plug-in* pour ce point d'extension nécessite tout d'abord de déployer toute notre architecture de métamodélisation sur un nouveau SGBD, puis, d'implémenter, pour ce nouveau SGBD, les différentes API d'accès nécessaires pour manipuler les données de notre architecture. Aucune nécessité immé-

24. http://semanticweb.org/wiki/OWL_API

diate de portabilité de la plate-forme DaFOE vers un nouvel environnement de persistante n'ayant été identifiée pour l'instant, ce point d'extension n'a pour l'instant été validé que sur un sous-ensemble réduit des modèles de la plat-forme DaFOE. La prise en compte de tous les modèles de la plate-forme DaFOE aurait nécessité plus de temps de développement.

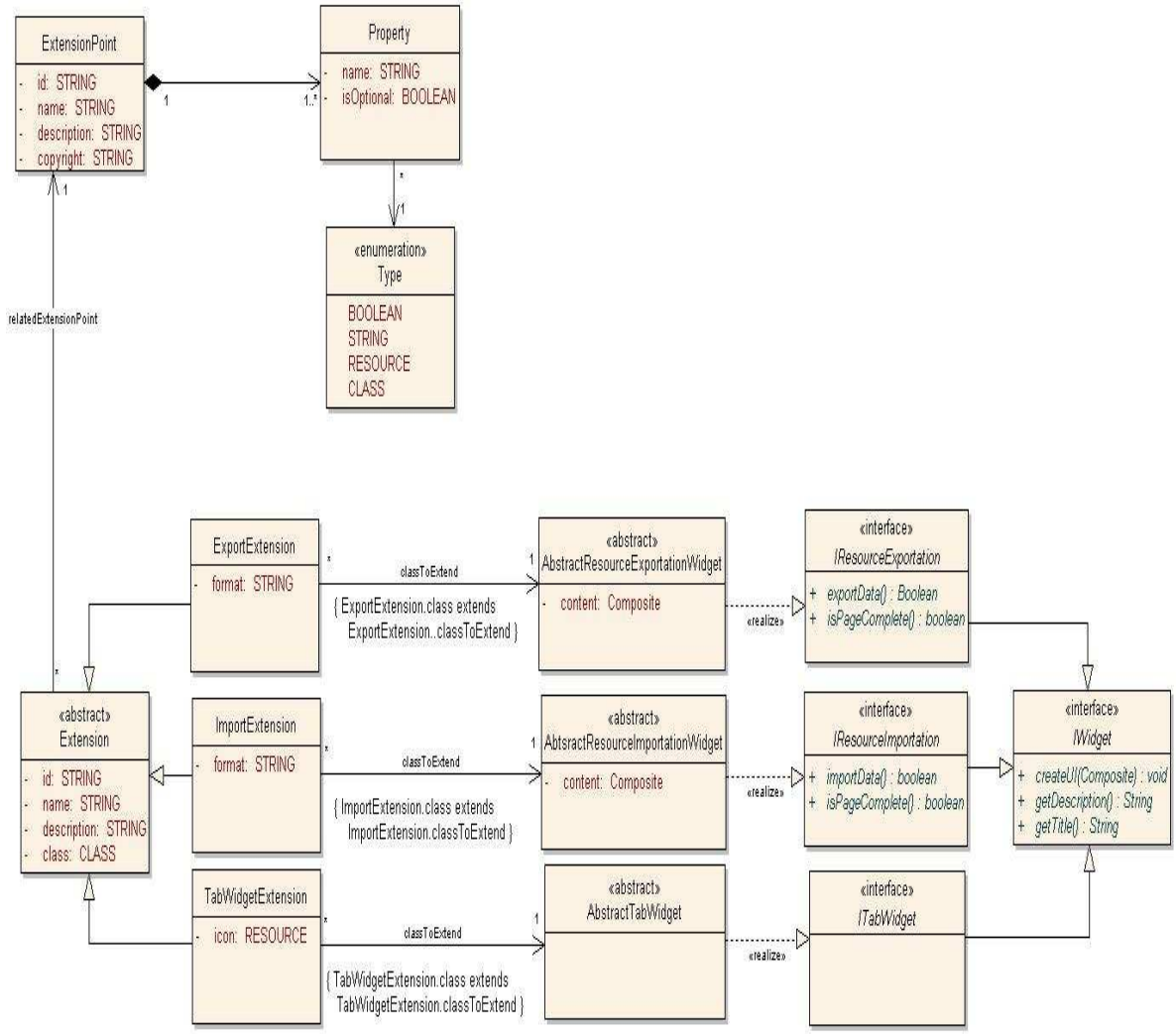

FIGURE 6.8 – Modélisation des points d'extension et des extensions.

6.5 Visite guidée de la plate-forme DaFOE

Dans cette section, nous présentons quelques éléments nécessaires à compréhension de l'utilisation de la plate-forme DaFOE. Nous abordons, d'une part, les aspects liés à l'espace de travail d'un utilisateur de la plate-forme, et, d'autre part, nous présentons le découpage modulaire de la plate-forme. Les figures illustrant cette section proviennent de l'utilisation de la plate-forme DaFOE pour la construction d'une ontologie à partir d'un corpus issu du domaine du patrimoine. Nous présentons notamment, l'apport

de MQLToolKit (grâce à la création des mappings) pour la semi-automatisation de cette construction d'ontologies.

Comme nous l'avons vu tout au long de cette thèse, le processus de construction d'ontologies à partir de corpus de textes, tel que proposé par la plate-forme DaFOE comporte trois étapes principales : une étape terminologique dédiée à l'analyse linguistique de corpus, une étape termino-ontologique dédiée à la désambigüisation des termes et relations issus du corpus et une étape ontologique permettant de créer les classes et propriétés de l'ontologie. Dans la plate-forme DaFOE, chaque étapes est accessible par un onglet. Un onglet correspond à un espace de travail permettant de manipuler les données de l'étape sélectionnée. Nous avons vu qu'un modèle est associé à chaque étape du processus de construction. La figure 6.9 illustre l'utilisation de MQLToolkit pour la création des différents modèles de la plate-forme DaFOE.

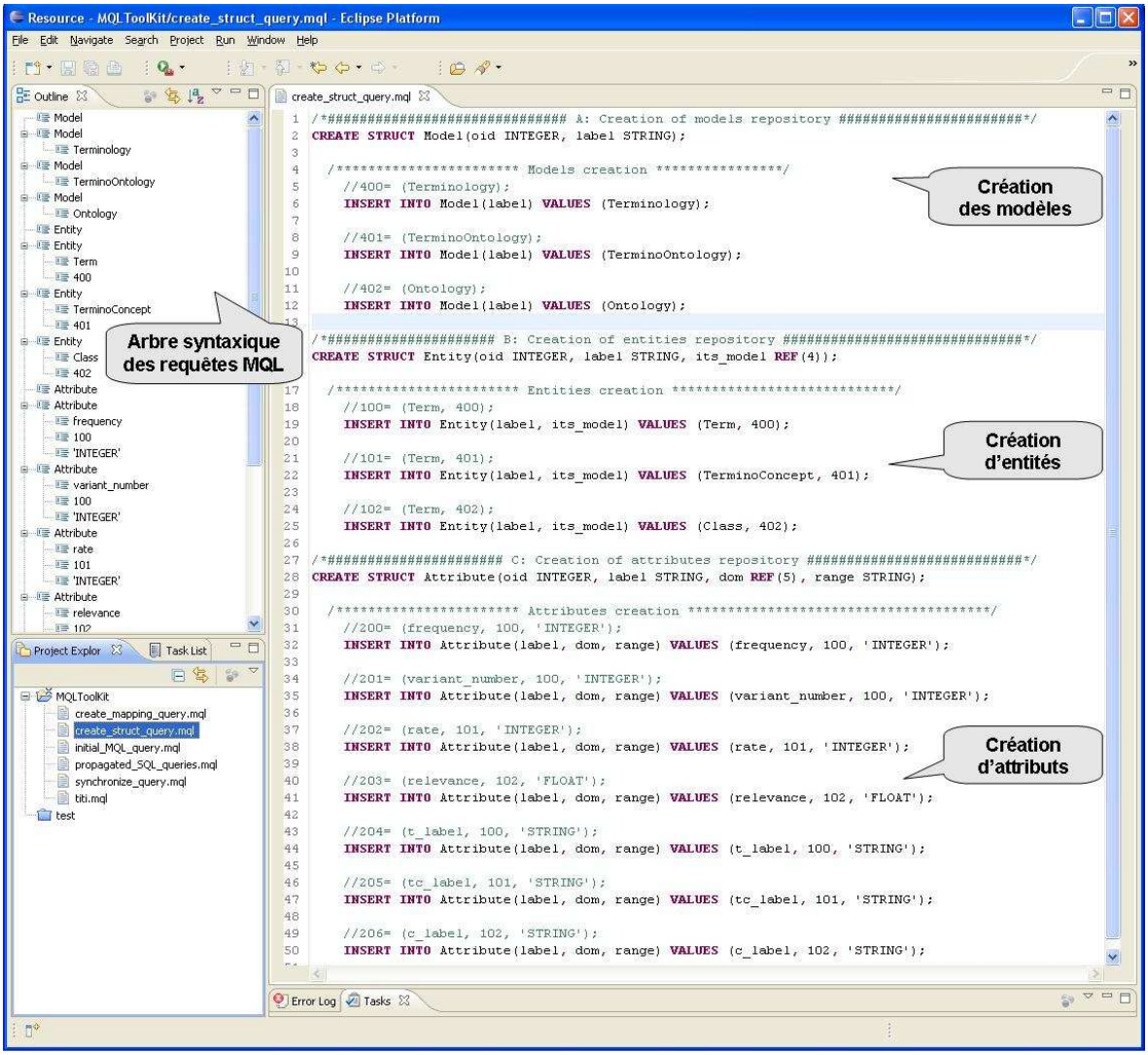

FIGURE 6.9 – Création de modèles avec MQLToolkit.

6.5.1 Onglet corpus

Bien que n'étant associé à aucune étape suivant la démarche de construction que nous avons présentée jusqu'ici, l'onglet *corpus* a néanmoins été introduit pour alléger les responsabilités (c'est-à-dire les types d'opération réalisables) dans l'étape terminologique. Cet onglet s'intéresse à l'affichage des corpus. Un corpus est constitué d'un ou de plusieurs documents. Cette étape fournit une visualisation du corpus phrase par phrase (cf. figure 6.10). Cette visualisation peut être alignée sur un terme, c'est-à-dire que, pour un terme sélectionné, on affiche, non pas toutes les phrases du corpus, mais uniquement les phrases dans lesquelles le terme apparaît. Toutefois, quel que soit le mode de visualisation, *l'allumage du terme* (mise en surbrillance des occurrences du terme dans le document sélectionné dans le corpus) permet de visualiser le terme dans ses différents contextes. Cette étape permet également d'importer un corpus au sein de la plate-forme DaFOE. Lors de l'importation d'un corpus, les termes du corpus sont automatiquement insérés dans l'étape terminologique.

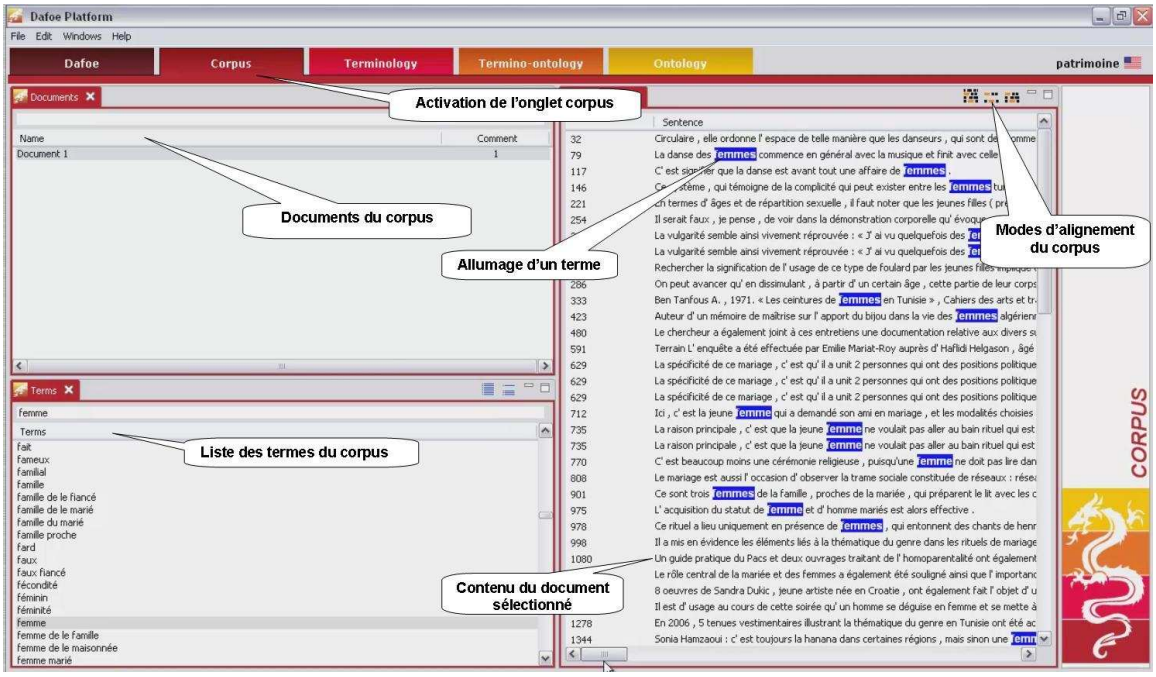

FIGURE 6.10 – Visualisation d'un corpus.

6.5.2 Onglet terminologique

L'étape terminologique est utilisée si la construction de l'ontologie exploite les résultats d'un outil de TAL ou si on a besoin de saisir des informations terminologiques associées aux *termino-concepts* ou aux relations *termino-conceptuelles* de l'étape termino-ontologique. En effet, la plate-forme DaFOE peut être utilisée à la fois pour construire une ontologie à partir de textes mais également pour construire une terminologie à partir d'une ontologie. Un *candidat terme* est considéré comme un *terme* dès lors qu'il est validé. Dans la plate-forme DaFOE, un *candidat terme* est représenté et interprété comme un terme en étude (*"studied"*). On parle dans ce cas de *statut d'un terme*. Un terme conceptualisé (dont le

statut est *"conceptualized"*) est un terme pour lequel un *termino-concept* a été associé. Dans l'onglet terminologique, trois types d'objets sont manipulés :

- les termes parmi lesquels on distingue les *entités nommées* identifiant, par exemple, des noms propres ;
- les relations reliant les termes ;
- les clusters de termes qui sont des ensembles de mots ou termes regroupés par proximité de sens.

Ces objets sont visualisables selon deux modes.

1. **Mode critère de saillance**
   Le *mode critère de saillance* (Cf. figure 6.11) propose l'affichage des termes ainsi que leurs informations d'analyse distributionnelle et statistique. Les termes peuvent être filtrés selon leur statut et des termes synonymes peuvent être regroupés autour d'un terme appelé *terme vedette*. Dans le corpus du domaine du patrimoine par exemple, les termes *"marié"* et *"époux"* sont synonymes et le terme *"marié"* a été choisi comme terme *vedette*.

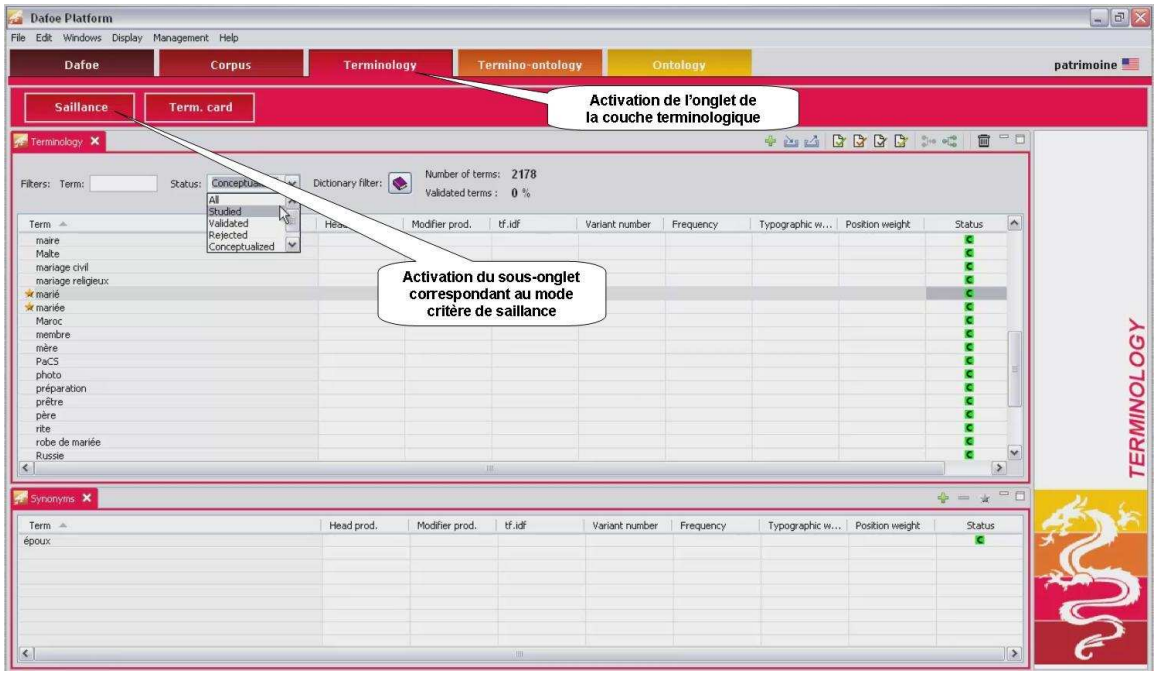

FIGURE 6.11 – Visualisation des termes en mode *critère de saillance*.

2. **Mode fiche terminologique**
   Le *mode fiche terminologique* (Cf. figure 6.12) permet notamment de consulter un terme ainsi que son interaction avec les autres termes à travers les différentes relations dans lesquelles le terme est impliqué. Une traçabilité permettant de visualiser les termino-concepts produits par le terme considéré est également proposée.

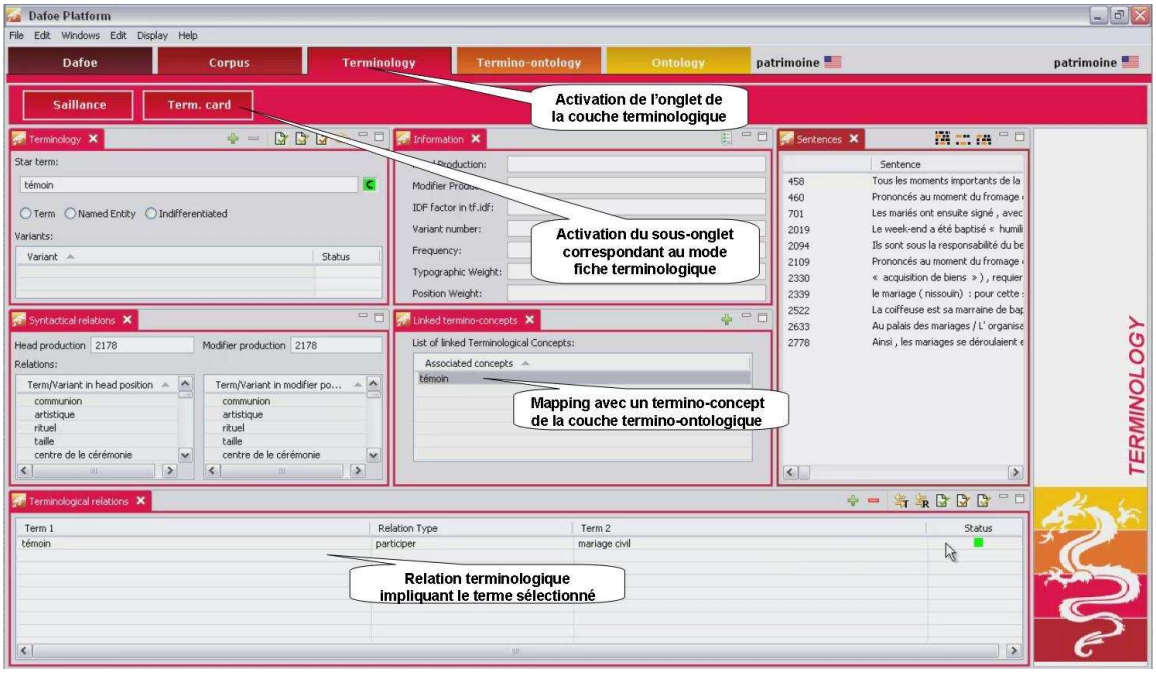

FIGURE 6.12 – Visualisation des termes en mode *fiche terminologique*.

6.5.3 Onglet termino-ontologique

Cet onglet permet de manipuler les données de l'étape termino-ontologique. Trois principales raisons justifient l'utilisation de cet onglet :

- on a exploité, dans l'étape terminologique, un outil d'analyse de corpus qui a fourni des informations terminologiques que l'on souhaiterait ensuite retraiter en vue d'une modélisation conceptuelle ;
- on récupère directement, dans l'étape termino-ontologique, une ressource de type thésaurus à partir de laquelle on veut construire une nouvelle ressource ;
- on souhaite ajouter une composante terminologique à une ontologie acquise dans l'étape ontologique.

Dès lors que les modèles terminologique et termino-ontologique ont été créés, MQLToolKit peut être utilisé, comme le montre la figure 6.13, pour créer des mappings. La synchronisation de l'entité `TerminoConcept` comparativement à l'entité `Term` (Cf. figure 6.14), par exemple, permet de produire automatiquement, à partir des *termes* de l'étape terminologique, des *terminoconcepts* de l'étape termino-ontologique (Cf. figure 6.15). Cette synchronisation montre, d'une part, que les *candidats termes* (car le statut doit être *"validated"*) ne seront par utilisés, et, d'autre part, que seuls les termes dont la *fréquence* est supérieure à `10` seront considérés. L'utilisation des clauses `With closure` et `DEPTH`, propage la synchronisation avec l'entité `Class` du modèle ontologique (en vertu de l'appariement existant entre l'entité `Class` et l'entité `TerminoConcept`). L'utilisateur peut également, relier manuellement les *terminoconcepts* produits à des *classes* de l'étape ontologique.

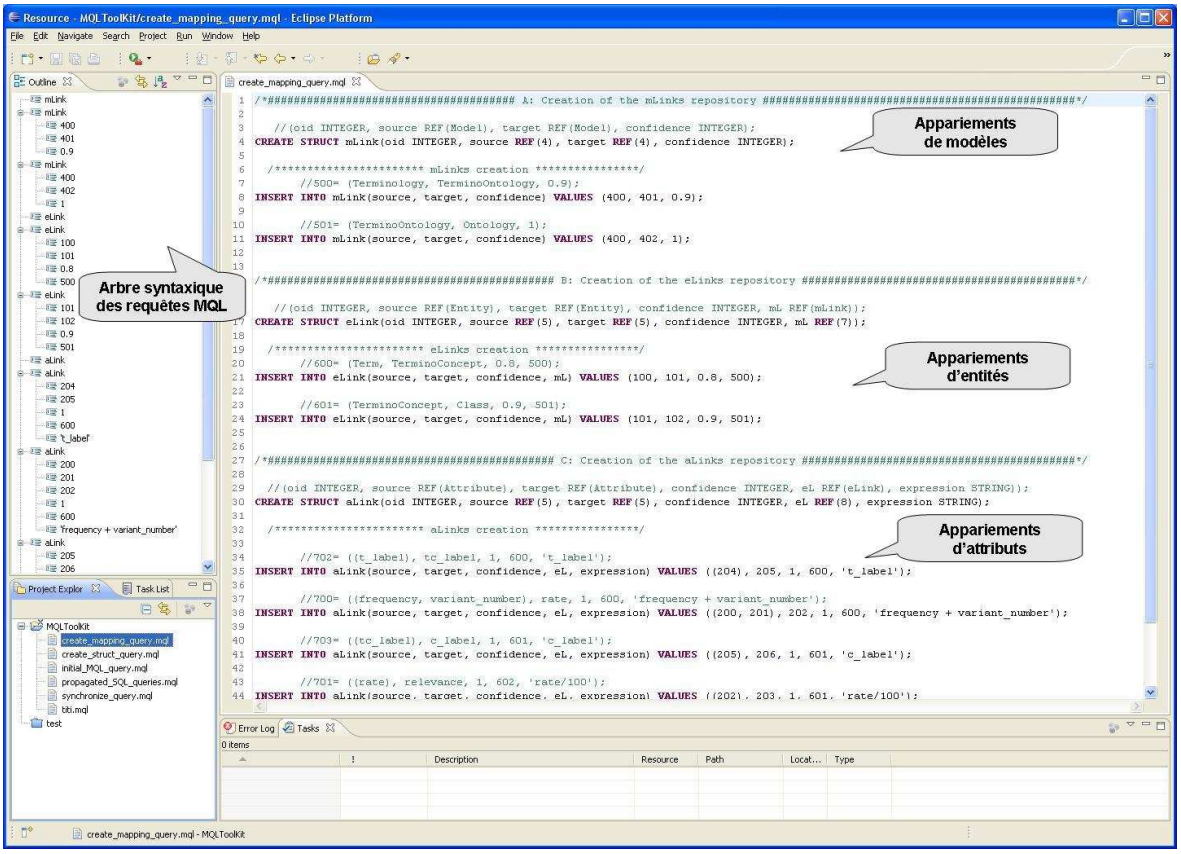

FIGURE 6.13 – Création des mappings avec MQLToolkit.

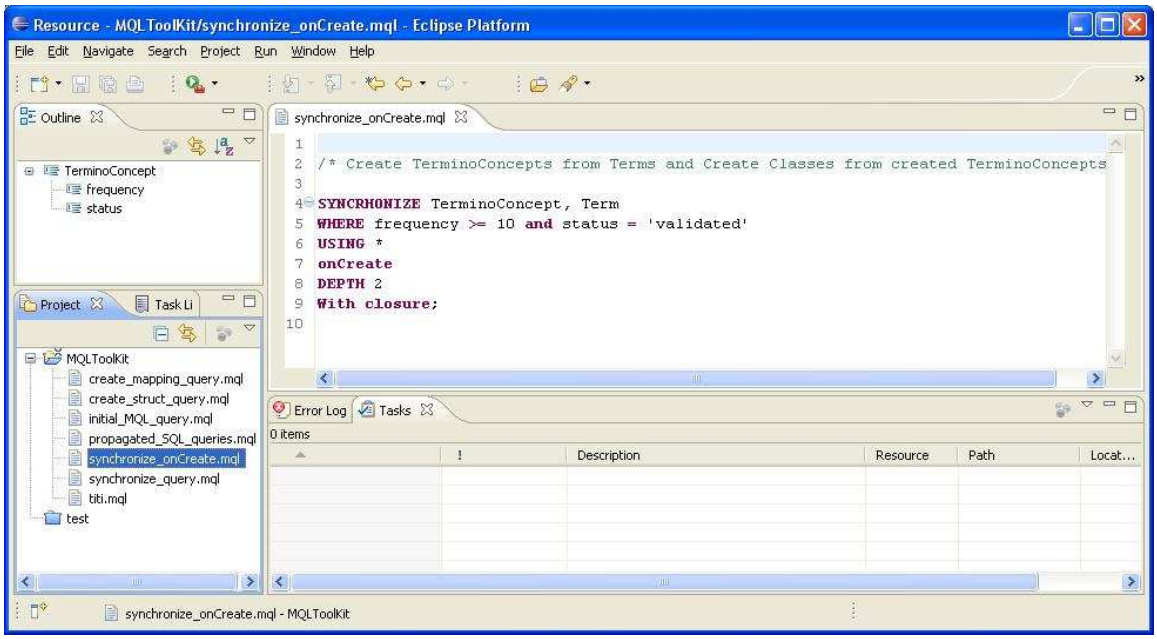

FIGURE 6.14 – Synchronisation des entités `Term` et `TerminoConcept`.

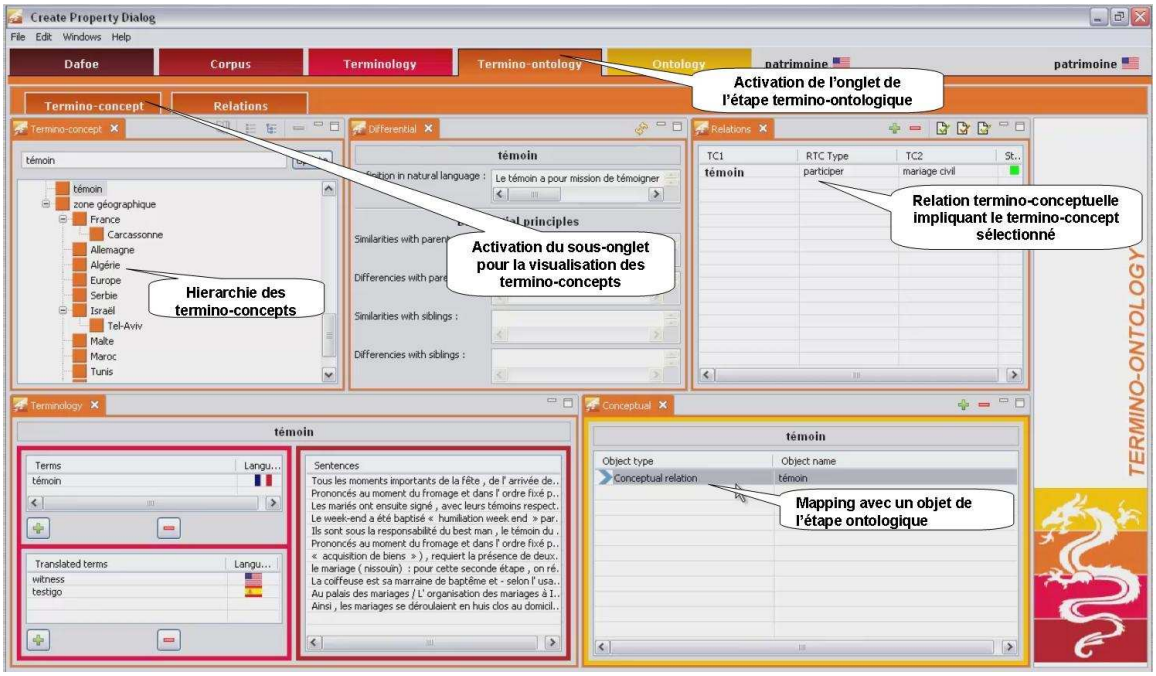

FIGURE 6.15 – Visualisation des termino-concepts.

6.5.4 Onglet ontologique

Cet onglet permet de manipuler les données de l'étape ontologique et correspond à l'étape finale du processus de construction d'ontologies. On y accède soit en provenance de l'étape termino-ontologique où une termino-ontologie a été construite, soit en important une ontologie existante qu'on souhaite enrichir. Trois sous-onglets permettent respectivement de visualiser des classes et leurs usages (Cf. figure 6.16), de visualiser des propriétés (Cf. figure 6.17) et des instances de classes. Quel que soit l'objet visualisé dans la couche ontologique, la plate-forme assure une traçabilité de la provenance de l'objet, c'est-a-dire qu'elle offre la possibilité de visualiser les objets de l'étape termino-ontologique ayant motivé la création de cet objet.

La figure 6.18 illustre la synchronisation de l'entités `Class` comparativement à l'entité `Termino-Concept` suite à une modification de l'appariement entre le rendement d'un termino-concept (représenté par l'attribut `rate`) et la pertinence d'une classe (représenté par l'attribut `relevance`). La fonction `sum` correspond à la somme des valeurs d'une colonne.

Comme nous l'avons précisé tout au long de ce mémoire, les mappings peuvent également être utilisés lors de l'interrogation des entités. La figure 6.19 illustre l'utilisation des mappings pour l'interrogation de l'entité `Class`. La requête MQL est transformée en plusieurs requêtes SQL qui sont ensuite exécutées.

Toutes ces fonctionnalités de la plate-forme DaFOE ont été utilisé par les partenaires du projet DaFOE4App tels que :

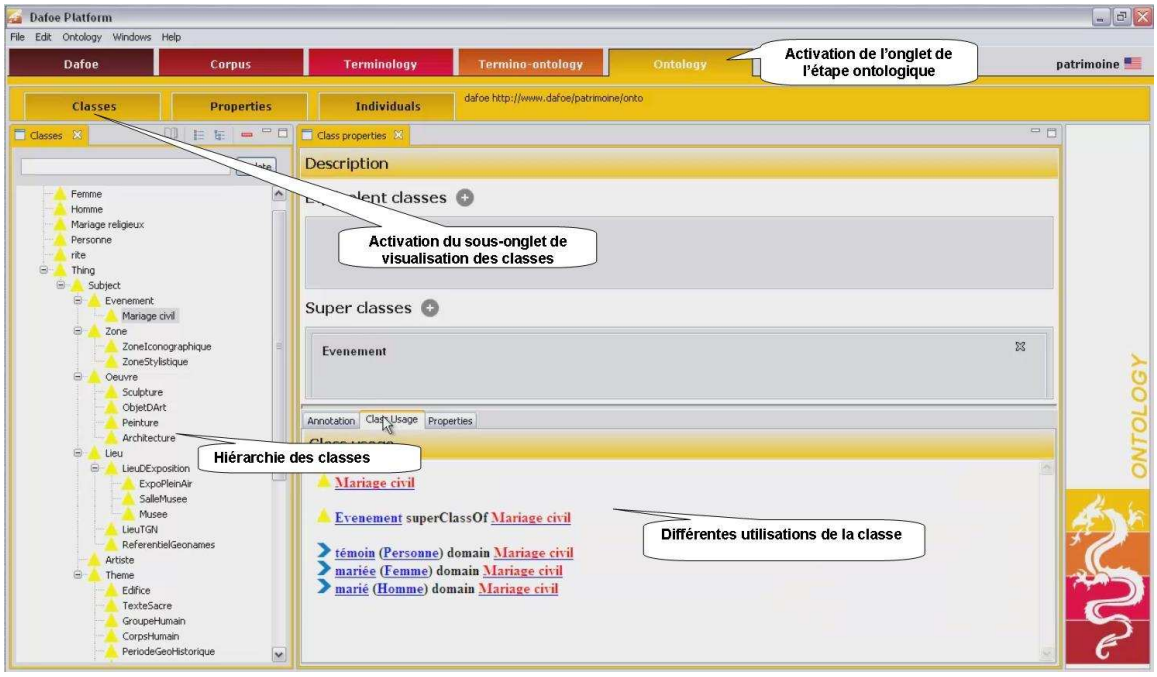

FIGURE 6.16 – Visualisation des classes d'une ontologie et leurs usages.

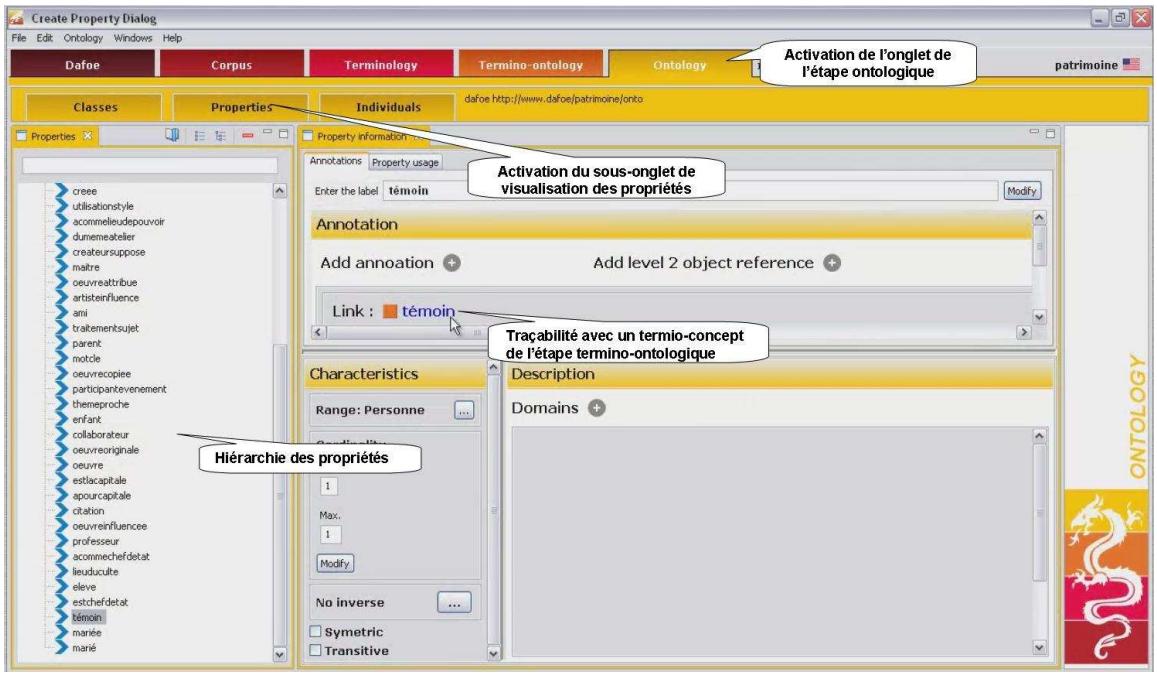

FIGURE 6.17 – Visualisation des propriétés d'une ontologie.

– le laboratoire LIPN [25], pour l'importation et exploitation de l'ontologies du domaine de la planifications des vols (*American Airlines AAdvantage*). Une ontologie de 223 classes et 24 propriétés a été construite à partir d'un corpus (d'environ 3000 termes) annoté par la plate-forme de TAL

25. http://lipn.univ-paris13.fr/fr/

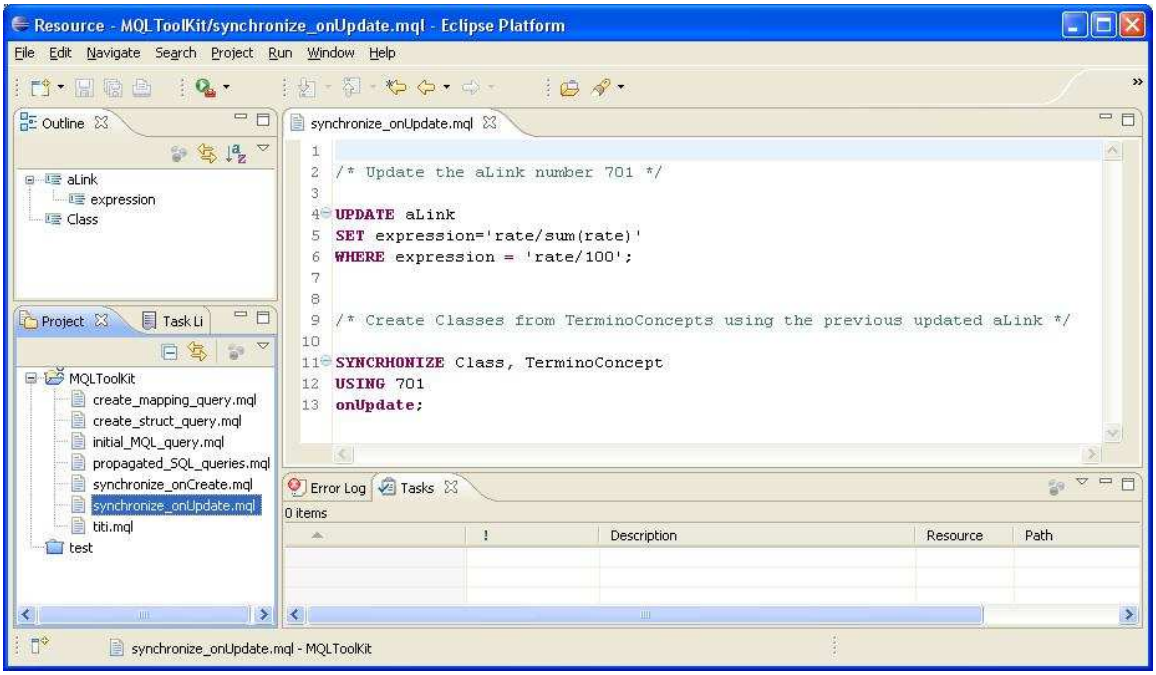

FIGURE 6.18 – Synchronisation des entités `TerminoConcept` et `Class`.

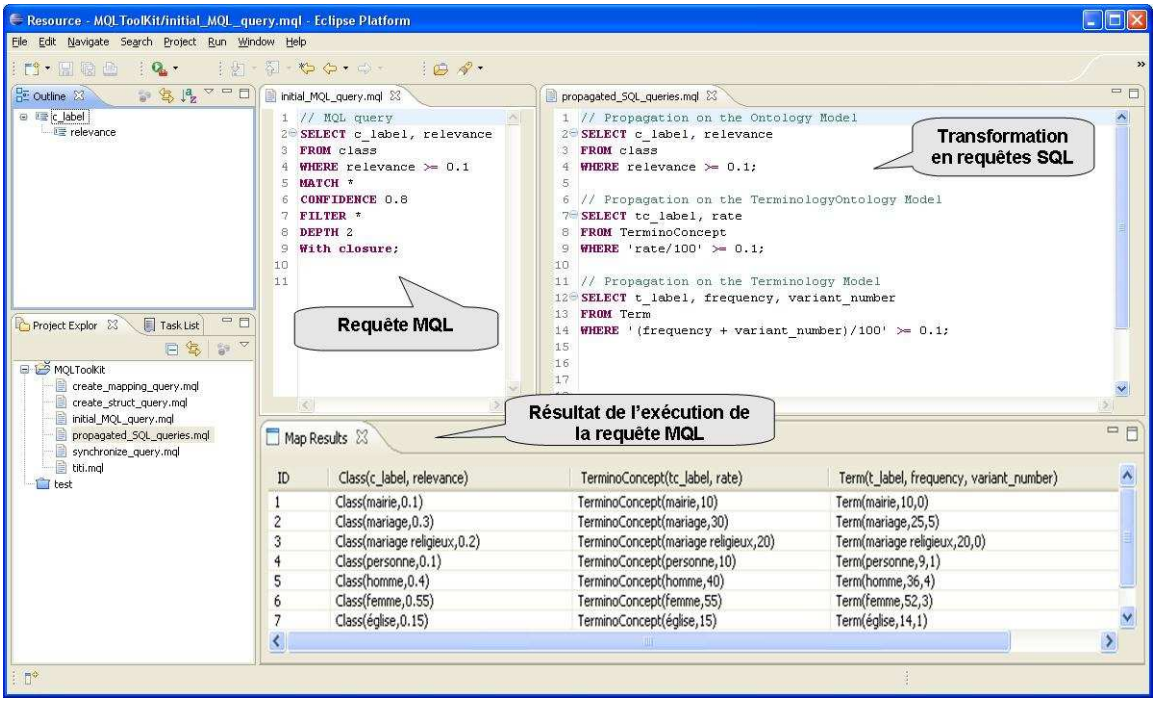

FIGURE 6.19 – Exploitation des mappings lors de l'interrogation.

Yatea ;

– la société Mondeca [26], dans le cadre de la construction d'une ontologie du patrimoine à partir d'un

26. http://www.mondeca.com/

corpus (d'environ 100.000 termes) annoté par la plate-forme de TAL Yatea.
– l'INSERM [27], pour la modélisation des connaissances médicales en pneumologie ;
– etc.

6.6 Conclusion

Dans ce chapitre, nous avons présenté le cadre technique de réalisation de la plate-forme DaFOE. Nous avons notamment rappelé les exigences techniques de la plate-forme DaFOE. L'une des exigences fondamentales étant le besoin d'extensibilité fonctionnelle, nous avons conçu et implémenté la plate-forme DaFOE selon une architecture à base de *plug-ins*, en utilisant la technologie Eclipse Rich Client Platform. Pour doter la plate-forme DaFOE de cette capacité d'extensibilité, nous avons identifié et modélisé quatre principales catégories de besoins en extension de la plate-forme DaFOE. Ainsi, la plate-forme DaFOE peut être étendue pour proposer de nouvelles entrées/sorties des données, pour proposer de nouvelles IHM ou encore pour proposer un nouvel environnement pour la persistance des données (changement de SGBD par exemple). Pour valider la capacité d'extension de la plate-forme DaFOE, de nombreux *plug-ins*, développés pour la plate-forme, ont été présentés dans ce chapitre. Conformément aux besoins de traçabilité du processus de construction au sein de la plate-forme DaFOE, nous avons proposé, pour chaque onglet permettant de manipuler les données des différentes étapes du processus de construction d'ontologies, des vues permettant, pour une donnée d'une étape, de visualiser les éventuels données auxquels elle est liée dans les autres étapes. Pour l'étape terminologique, par exemple, et pour un *terme* donné, il est possible de visualiser les éventuels *terminoconccepts* produits par ce *terme*. De la même manière, dans l'étape ontologique, pour une *classe* donnée, il est possible de visualiser les éventuels *terminoconcepts* dont la classe est issue. Sur le plan des mappings, l'outil MQLToolkit, réalisé en utilisant le *framework* Xtext et disponible sous forme de plug-in pour la plate-forme DaFOE, offre un environnement d'édition et d'exécution des requêtes MQL.

27. http://www.inserm.fr/

Conclusion et Perspectives

"In theory, there is no difference between theory
and practice. But, in practice, there is."
– Jan L.A van de Snepscheut

Conclusion

Le travail que nous avons présenté dans cette thèse s'inscrivait dans le contexte du projet DaFOE4App dont les principaux objectifs étaient de :

- $\mathbf{O}_1$) définir un cadre méthodologique général susceptible d'intégrer la réalisation des scénarios très divers de conception d'ontologies ;
- $\mathbf{O}_2$) définir une structure de modélisation permettant de s'adapter à ces différents scénarios de conception ;
- $\mathbf{O}_3$) spécifier puis développer une plate-forme (la plate-forme DaFOE) capable d'accueillir et d'intégrer les différentes catégories d'outils actuellement utilisés de façon autonomes au sein d'une structure de modélisation unique, assurant à la fois persistance et traçabilité, et permettant, par exemple, d'aller de l'analyse d'un corpus, supposé annoté par une plate-forme de traitement automatique de la langue (TAL), à la définition d'une ontologie formelle de domaine.

Dans la première partie de ce mémoire, nous avons tout d'abord présenté une vue globale des approches existantes pour la construction d'ontologies, puis nous avons détaillé les approches qui, comme celle que nous avons présentée dans cette thèse, utilisent des textes comme connaissances à priori. Nous avons évoqué que notre approche se distingue des autres à la fois par l'utilisation des transformations de modèles, mais aussi par la manière dont nous modélisons ces transformations. Ainsi, la contribution de ce travail se résume selon trois principaux axes :

❶ la modélisation à base de mappings d'une méthodologie de construction d'ontologies ;

❷ la persistance des mappings ;

❸ l'exploitation des mappings.

Modélisation à base de mappings d'une méthodologie de construction d'ontologies

Ayant constaté que la diversité des domaines des différents acteurs (ingénierie de la connaissance, linguistique, expert du domaine à modéliser, ontologue, etc.) impliqués dans la démarche de construction d'ontologies à partir de textes, rendait difficile la représentation, au sein d'un unique modèle, de la méthodologie de construction d'ontologies, nous avons proposé, d'une part, de modéliser de manière quasi-autonome chacune des étapes de cette méthodologie de construction, et, d'autre part, d'utiliser des mappings pour décrire le passage d'une étape à l'autre. Cette séparation des modèles associés aux différentes étapes a également été motivée par le besoin d'évolution des modèles compte tenu de l'hétérogénéité des sources de données (outil TAL par exemple) utilisées pour alimenter la plate-forme DaFOE.

En nous positionnant en faveur des approches par méta-modèle pour la représentation des mappings, nous avons distingué trois principaux types d'appariements suivant la nature des éléments (modèle, entité, attribut) à apparier. Toutefois, nous avons montré que notre approche ne se limitait pas à ces principaux types d'appariements. En effet, notre approche permet non seulement de créer de nouveaux types d'appariements mais également de modifier les constructeurs existants grâce à la disposition d'un méta-méta-modèle. Constatant que la composition est une opération fondamentale pour les mappings, nous avons, dans cette thèse, caractérisé la composition pour les types d'appariements proposés. Cette caractérisation ne représente qu'un calcul par défaut, un utilisateur pouvant faire évoluer le modèle de mappings. Afin d'adapter cette caractérisation à une éventuelle évolution du modèle de mappings, nous avons modélisé les fonctions de composition au sein du méta-méta-modèle. En d'autres termes, sur le plan de la représentation des mappings, à la différence des travaux proposés dans la littérature, notre travail propose une approche générique dans laquelle la représentation des mappings est dynamiquement extensible. Par conséquent, si la méthodologie de construction d'ontologies présentée dans cette thèse comporte à la base trois étapes principales, nous l'avons généralisée à une méthodologie pouvant comporter un nombre quelconque d'étapes car un utilisateur peut la raffiner en y ajoutant, supprimant ou modifiant une ou plusieurs étape(s). Un raffinement consistant dans ce cas à la création, suppression ou modification de modèle(s), suivie d'une configuration adéquate des mappings.

Persistance des mappings

Pour les besoins de la plate-forme DaFOE, et compte tenu du volume de données à manipuler, nous avons choisi de stocker les données dans une base de données. Il fallait donc proposer un mécanisme de persistance du processus de construction d'ontologies. En d'autres termes nous avions besoin de conserver au sein d'une base de données, à la fois, les modèles représentant les différentes étapes de la méthodologie de construction d'ontologies, les mappings entre ces modèles ainsi que les instances de ces modèles. Une étude préliminaire des BDBM nous a permis de montrer que, si ce modèle d'architecture de base de données était mieux adapté à la persistance, au sein d'une unique base de données, des données et des méta-données, les différents BDBM proposées dans la littérature ne prenaient cependant pas en compte la représentation des mappings. Il était donc difficile de persister le processus de construction d'ontologies telle que nous la modélisions. Nous avons de ce fait proposé une adaptation des BDBM de manière à supporter le stockage des mappings. Ce stockage des mappings, s'appuyant sur un méta-méta-modèle conformément à notre proposition de méta-modélisation, permet, d'une part, la création et la modification de nouveaux constructeurs de mappings, et, d'autre part, la génération automatique de la

structure physique pour la persistance des mappings.

Exploitation des mappings

Dans une application à base de transformation de modèles, une fois que les modèles et les mappings sont créés et instanciés dans la BDBM, il est parfois intéressant, par exemple, d'exploiter ces mappings lors de l'interrogation des données. Dans le cas du processus de construction d'ontologies que nous avons présenté dans cette thèse, les mappings sont utilisés pour mettre en correspondance des modèles hétérogènes modélisant le même domaine mais avec des niveaux de formalisation différents. Pour le processus de recherche d'informations dans un modèle qui pourrait exploiter les mappings, nous avons identifié l'écriture des requêtes, l'exploration du graphe de mappings, la saturation mémoire et la cohérence des instances comme des problèmes posés pour l'exploitation des mappings pour une application modélisée en utilisant le mapping de modèles dans un environnement persistant de base de données. L'exploitation des mappings dans une base de données faisant référence à la nécessité de disposer d'un mécanisme d'interrogation de la base de données en utilisant les mappings, nous avons évoqué que les différents langages proposés dans la littérature ne proposaient pas de solution à ces différents problèmes. Par conséquent, pour les applications basées sur le mapping de modèles comme c'est le cas de la plate-forme DaFOE que nous avons présentée tout au long de cette thèse, nous avons proposé le langage de requête MQL. Le langage MQL s'appuie sur l'architecture de méta-modélisation proposée par les BDBM et permet de manipuler les données (instances) et méta-données (modèles et méta-modèle).

Pour la manipulation des méta-données, le langage MQL propose un sous-ensemble fonctionnel permettant de créer, supprimer ou modifier des éléments des modèles (entités, attributs, etc.) et des méta-modèles (méta-entités, méta-attributs, etc.). L'originalité du langage MQL réside dans sa capacité à permettre d'étendre le méta-modèle.

Du point de vue de la manipulation des données, le langage MQL propose un sous-ensemble fonctionnel dédié à l'accès des données au niveau logique, permettant ainsi de manipuler les données comme dans une base de données relationnelle en restant compatible avec le langage SQL implanté par les SGBD couramment utilisés. De plus, pour l'accès aux données au niveau logique, le langage MQL propose une extension du langage SQL permettant la manipulation conjointe des données et des méta-données. Cette extension se traduit par la présence, dans la syntaxe du langage MQL, des clauses optionnelles permettant, lors de l'accès à des données, de spécifier des contraintes sur des méta-données. C'est le cas par exemple, dans la plate-forme DaFOE, lorsqu'une requête sur le modèle ontologique est directement propagée sur le modèle termino-ontologique et transitivement propagée sur le modèle terminologique. L'originalité du langage MQL dans ce cas est double. Les clauses proposées par le langage MQL permettent, d'une part, à un utilisateur de paramétrer le mode propagation des requêtes. L'avantage d'un tel paramétrage est qu'il permet à un utilisateur de spécifier un sous-ensemble du graphe de mappings qu'il estime pertinent pour sa requête. D'autre part, le langage MQL fournit une syntaxe compacte facilitant l'exploration du graphe de mappings aussi bien pour l'interrogation des données que pour leurs manipulation.

Proposer un langage de requêtes de bases de données sans en donner la sémantique rend difficile son implantation, l'étude de ses propriétés sémantiques et l'optimisation des requêtes écrites dans ce langage. Nous avons donc défini une algèbre d'opérateurs nommée *MapAlgebra*. *MapAlgebra* est une

algèbre qui s'inspire de l'algèbre multi-relationnelle proposée pour les bases de données fédérées ou multibases en y ajoutant la représentation explicite des mappings. Le langage MQL implémente de ce fait tous les opérateurs proposé par *MapAlgebra*. Du point de vue implémentation, les requêtes MQL sont, à l'exécution, traduite en une ou plusieurs requête(s) SQL exécutable(s) sur le SGBD. Pour garantir cette traduction MQL vers SQL, nous avons montré, pour chaque opérateur de *MapAlgebra* sa décomposition en opérateur(s) de l'algèbre relationnelle sur laquelle s'appuie le langage SQL. L'existence d'une telle sémantique algébrique rend générique notre approche car en cas de changement d'environnement de persistance (vers des fichiers XML par exemple), il est possible d'adapter le langage MQL en implémentant des opérateurs de *MapAlgebra* mais cette fois suivant la structure de fichier XML en utilisant un transduction vers le langage XQuery [28] par exemple.

Réalisation

Les différentes contributions de cette thèse ont aidé à la réalisation de la plate-forme DaFOE, dédiée à la construction d'ontologies dont l'une des exigences fondamentales portait sur le besoin d'extensibilité fonctionnelle de la plate-forme DaFOE. Comme solution à cette exigence technique, nous avons conçu et implémenté la plate-forme DaFOE selon une architecture à base de plug-ins en utilisant la technologie Eclipse Rich Client Platform. Pour doter la plate-forme de cette capacité d'extensibilité, nous avons identifié et modélisé quatre catégories de besoins en extension de la plate-forme DaFOE. Ainsi, la plate-forme DaFOE peut être étendue pour proposer de nouvelles entrées/sorties des données, pour proposer de nouvelles IHM ou encore pour proposer un nouvel environnement pour la persistance des données (changement de SGBD par exemple). Conformément aux besoins de traçabilité du processus de construction au sein de la plate-forme DaFOE, nous avons proposé, pour chaque étapes du processus de construction d'ontologies, des vues permettant de manipuler les données de l'étape tout en affichant, pour chaque donnée, une visualisation des éventuelles données auxquelles elle est liée dans les autres autres étapes. Pour la couche terminologique par exemple, et pour un *terme* donné, il est possible de visualiser les éventuels *terminoconcepts* produits par ce *terme*. De la même manière, dans la couche ontologique, pour une *classe* donnée, il est possible de visualiser les éventuels *terminoconcepts* dont la classe est issue.

L'outil MQLToolkit, réalisé en utilisant le framework Xtext et disponible sous forme de plug-in pour la plate-forme. Il offre un environnement d'édition et d'exécution des requêtes MQL.

Perspectives

Les travaux présentés dans ce mémoire laissent envisager de nombreuses perspectives aussi bien sur les performances du système résultant que sur le langage d'exploitation des mappings.

28. http://www.w3.org/TR/xquery/

Tests de performances

Dans notre travail de thèse, nous n'avons pas abordé les aspects liés à l'évaluation des performances de la structure de stockage, car nous avons fait l'hypothèse selon laquelle *une BDBM étendue pour la prise en compte des mappings demeure une BDBM*. Ce qui signifie que, nous admettons que les différents résultats de tests de performances réalisés pour les BDBM restent valables en dépit de l'extension que nous avons apportée. Si de prime à bord il nous semble que cette hypothèse serait toujours vérifiée, nous pensons que seule une analyse de performance réelle pourrait davantage valider cette hypothèse. Dans ce cas, dans la continuité de ce travail, nous envisageons de décrire des scénarios d'évaluations de performances de la solution de stockage de mappins que nous avons proposée. Il s'agira notamment d'évaluer le coût des opérations transitives des mappings. En fonction des résultats obtenus, la question de l'utilisation des structures d'optimisation proposées par les travaux sur les bases de données pourraient à cet effet être éventuellement envisagée.

Optimisation des requêtes MQL

Au démarrage de cette thèse, nous n'avions pas à l'esprit de concevoir un langage de requête. Mais, au fur et à mesure de l'évolution du travail, la nécessité d'un tel langage est apparue notamment lorsqu'il s'est posé le besoin de simplifier l'exploitation des mappings. Même si les résultats obtenus avec le langage MQL dans cadre la plate-forme DaFOE nous paraissent satisfaisants, nous pouvons néanmoins dire que le langage MQL est encore dans une phase "d'adolescence", et, pour arriver à maturité, le langage MQL mériterait d'être évalué d'un point de vue optimisation de requêtes. Le travail qui pourrait être fait à cet effet consisterait, étant donné que nous avons montré l'existence d'une interprétation des expressions du langage MQL par des expressions du langage SQL (par décomposition des opérations de *MapAlgebra* en fonction de celles de l'algèbre relationnelle), de préciser les techniques d'optimisations du langage MQL en s'appuyant sur celles de l'algèbre relationnelle. Une fois l'étude de ces techniques d'optimisation faite, nous saurons, par exemple, si la syntaxe compacte (des requêtes), telle que proposée par le langage MQL, n'est pas dommageable pour la complexités algorithmes associés aux requêtes lors de la réécriture d'une requête MQL en terme de requête(s) SQL.

MQL, une solution possible pour l'intégration de données?

Le langage MQL que nous avons présenté dans cette thèse nous a permis de réaliser une interrogation multi-modèles à l'échelle d'une unique base de données comportant plusieurs modèles. Une perspective de notre travail consisterait, par exemple, à étendre le langage MQL au cas des systèmes distribués pour lesquels les modèles sont en général stockés dans des bases de données multi-sites. En d'autres termes, il s'agira d'envisager la possibilité d'utiliser le langage MQL dans un processus d'intégration de données à la volée. Pour les systèmes d'intégration de type Local As View par exemple, le langage MQL pourrait être adapté de manière à ce que l'interrogation d'une source de données puissent être propagée sur d'autres sources de données pour répondre au mieux à une requête d'un utilisateur. Pour cela, l'algèbre *MapAlgebra* ne devrait plus se limiter aux opérations satisfaisant aux besoins de la plate-forme DaFOE, mais devrait être enrichie de manière à garantir sa complétude.

Bibliographie

[Abadi et al., 2007] Abadi, D. J., Marcus, A., Madden, S. R., and Hollenbach, K. (2007). Scalable semantic web data management using vertical partitioning. In *Proceedings of the 33rd International Conference on Very Large Data Bases*, VLDB '07, pages 411–422. VLDB Endowment.

[Agrawal et al., 2001] Agrawal, R., Somani, A., and Xu, Y. (2001). Storage and querying of e-commerce data. In *Proceedings of the 27th International Conference on Very Large Data Bases*, VLDB '01, pages 149–158, San Francisco, CA, USA. Morgan Kaufmann Publishers Inc.

[Alexaki et al., 2001] Alexaki, S., Christophides, V., Karvounarakis, G., Plexousakis, D., and Tolle, K. (2001). The ics-forth rdfsuite: Managing voluminous rdf description bases. In *Proceedings of the 2nd International Workshop on the Semantic Web (SemWeb'01)*.

[An et al., 2006] An, Y., Borgida, A., and Mylopoulos, J. (2006). Discovering the semantics of relational tables through mappings. *Journal on Data Semantics VII*, pages 1–32.

[Aubin and Hamon, 2006] Aubin, S. and Hamon, T. (2006). Improving term extraction with terminological resources. In Salakoski, T., Ginter, F., Pyysalo, S., and Pahikkala, T., editors, *5th International Conference on NLP, FinTAL*, Lecture Notes in Computer Science, pages 380–387. Springer.

[Bachimont, 2000] Bachimont, B. (2000). *Engagement Sémantique et Engagement Ontologique : Conception et Réalisation D'ontologies En Ingénierie Des Connaissances*, chapter 19, pages 305–324.

[Barrasa et al., 2004] Barrasa, J., Corcho, Ó., and Gómez-Pérez, A. (2004). R2o, an extensible and semantically based database-to-ontology mapping language. In *Proceedings of the 2nd Workshop on Semantic Web and Databases (SWDB'04)*.

[Benjamins et al., 1999] Benjamins, V. R., Fensel, D., Decker, S., and Perez, A. G. (1999). (ka)2: building ontologies for the internet: a mid-term report. *International Journal of Human-Computer Studies*, 51(3):687–712.

[Berndtsson and Lings, 1995] Berndtsson, M. and Lings, B. (1995). Logical events and eca rules. Technical report, University of Skövde.

[Berners-Lee et al., 2001] Berners-Lee, T., Hendler, J., and Lassila, O. (2001). The Semantic Web. *Scientific American*, 284(5):34–43.

[Bernstein, 2003] Bernstein, P. A. (2003). Applying model management to classical meta data problems. In *Proceedings of the International Conference on Innovative Data Systems Research*.

[Bézivin and Gerbé, 2001] Bézivin, J. and Gerbé, O. (2001). Towards a precise definition of the omg/mda framework. In *Proceedings of the 16th IEEE international conference on Automated software engineering*, ASE '01, pages 273–

, Washington, DC, USA. IEEE Computer Society.

[Biebow and Szulman, 1999] Biebow, B. and Szulman, S. (1999). Terminae: A linguistic-based tool for the building of a domain ontology. In *Proceedings of the 11th European Workshop on Knowledge Acquisition, Modeling and Management*, EKAW '99, pages 49–66, London, UK. Springer-Verlag.

[Blomqvist, 2009] Blomqvist, E. (2009). Ontocase-automatic ontology enrichment based on ontology design patterns. In *Proceedings of the International Semantic Web Conference*, pages 65–80.

[Broekstra et al., 2002] Broekstra, J., Kampman, A., and van Harmelen, F. (2002). Sesame: A generic architecture for storing and querying rdf and rdf schema. In *International Semantic Web Conference*, pages 54–68.

[Charles Ogden, 1923] Charles Ogden, I. R. (1923). *The Meaning of Meaning*. Paul Kegan.

[Charlet et al., 2010a] Charlet, J., Szulman, S., Aussenac-Gilles, N., Nazarenko, A., Hernandez, N., Nada, N., Sardet, E., Delahousse, J., Téguiak, H. V., and Baneyx, A. (2010a). Dafoe : une plate-forme pour construire des ontologies à partir de textes et de thésaurus (poster). In *Journées Francophones Extraction et Gestion de Connaissances*, EGC'10, pages 1–3.

[Charlet et al., 2010b] Charlet, J., Szulman, S., Aussenac-Gilles, N., Nazarenko, A., Valéry, T., and Eric, S. (2010b). Dafoe: A platform for building ontologies from texts (poster and demo). In *17th International Conference on Knowledge Engineering and Knowledge Management*, EKAW'10, pages –.

[Charlet et al., 2008] Charlet, J., Szulman, S., Pierra, G., Nadiah, N., Téguiak, H. V., Aussenac-Gilles, N., and Nazarenko, A. (2008). Dafoe: A multimodel and multimethod platform for building domain ontologies. In *Actes des 2ièmes Journées Francophones sur les Ontologies (JFO'2008)*, pages 74–85.

[Chawathe et al., 1994] Chawathe, S., Garcia-Molina, H., Hammer, J., Ireland, K., Papakonstantinou, Y., Ullman, J. D., and Widom, J. (1994). The tsimmis project: Integration of heterogeneous information sources. In *Meeting of the Information Processing Society of Japan*, pages 7–18.

[Chen, 1976] Chen, P. P.-S. (1976). The entity-relationship model - toward a unified view of data. *ACM Trans. Database Syst.*, 1(1):9–36.

[Chong et al., 2005] Chong, E. I., Das, S., Eadon, G., and Srinivasan, J. (2005). An efficient sql-based rdf querying scheme. In *Proceedings of the International Conference on Very Large Data Bases*, pages 1216–1227.

[Cimiano and Völker, 2005] Cimiano, P. and Völker, J. (2005). Text2onto - a framework for ontology learning and data-driven change discovery. In Montoyo, A., Munoz, R., and Metais, E., editors, *Proceedings of the 10th International Conference on Applications of Natural Language to Information Systems (NLDB'05)*, volume 3513 of *Lecture Notes in Computer Science*, pages 227–238, Alicante, Spain. Springer.

[Codd, 1970] Codd, E. F. (1970). A relational model of data for large shared data banks. *Commun. ACM*, 13(6):377–387.

[Dalvi and Suciu, 2007] Dalvi, N. and Suciu, D. (2007). Efficient query evaluation on probabilistic databases. *The VLDB Journal*, 16(4):523–544.

[Dean and Schreiber, 2004] Dean, M. and Schreiber, G. (2004). *OWL Web Ontology Language Reference*. World Wide Web Consortium. `http://www.w3.org/TR/owl-ref`.

[Dehainsala et al., 2007a] Dehainsala, H., Pierra, G., and Bellatreche, L. (2007a). Ontodb: an ontology-based database for data intensive applications. In *Proceedings of the 12th Interna-*

tional Conference on Database Systems for Advanced Applications, DASFAA'07, pages 497–508, Berlin, Heidelberg. Springer-Verlag.

[Dehainsala et al., 2007b] Dehainsala, H., Pierra, G., Bellatreche, L., and Aït-Ameur, Y. (2007b). Conception de bases de données à partir d'ontologies de domaine : Application aux bases de données du domaine technique. In *Actes des 1ère Journées Francophones sur les Ontologies (JFO'07)*, pages 215–230.

[del Mar Roldán García et al., 2005] del Mar Roldán García, M., Delgado, I. N., and Montes, J. F. A. (2005). A design methodology for semantic web database-based systems. In *Proceedings of the 3rd International Conference on Information Technology and Applications (ICITA'05)*, pages 233–237. IEEE Computer Society.

[Didier Bourigault and Ozdowska, 2005] Didier Bourigault, Marie-Paule Jacques, C. F. C. F. and Ozdowska, S. (2005). Syntex, analyseur syntaxique de corpus. In *12èmes journées sur le Traitement Automatique des Langues Naturelles*.

[Dietz, 2006] Dietz, J. L. G. (2006). *Enterprise ontology - theory and methodology*. Springer.

[Estival et al., 2004] Estival, D., Nowak, C., and Zschorn, A. (2004). Towards Ontology-based Natural Language Processing. In *RDF/RDFS and OWL in Language Technology: 4th Workshop on NLP and XML (NLPXML-2004), ACL 2004*.

[Euzenat, 1995] Euzenat, J. (1995). Building consensual knowledge bases: Context and architecture. In Mars, N., editor, *Towards Very Large Knowledge Bases - Proceedings of the KB&KS '95 Conference*, pages 143–155.

[Euzenat, 1996] Euzenat, J. (1996). Corporate memory through cooperative creation of knowledge bases and hyper-documents. In *Proceedings of the 10th Knowledge Acquisition, Modeling and Management for Knowledge-based Systems Workshop*, pages 1–18.

[Euzenat and Shvaiko, 2007] Euzenat, J. and Shvaiko, P. (2007). *Ontology matching*. Springer-Verlag, Heidelberg (DE).

[Fankam, 2009] Fankam, C. (2009). *OntoDB2 : un système flexible et efficient de Base de Données à Base Ontologique pour le Web sémantique et les données techniques*. PhD thesis, ENSMA, France.

[Fankam et al., 2008] Fankam, C., Pierra, G., and Bellatreche, L. (2008). Ontodb2: Support of multiple ontology models within ontology. In *Proceedings of the EDBT 2008 PhD Workshop*.

[Farquhar et al., 1995] Farquhar, A., Fikes, R., Pratt, W., and Rice, J. (1995). Collaborative ontology construction for information integration keywords. *Technique Reports of Knowledge Systems Laboratory Department of Computer Science KSL95*, 63.

[Foskett, 1997] Foskett, D. J. (1997). *Thesaurus*, pages 111–134. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA.

[Fuhr and Rölleke, 1997] Fuhr, N. and Rölleke, T. (1997). A probabilistic relational algebra for the integration of information retrieval and database systems. *ACM Trans. Inf. Syst.*, 15(1):32–66.

[Gangemi et al., 2003] Gangemi, A., Guarino, N., Masolo, C., and Oltramari, A. (2003). Sweetening wordnet with dolce. *AI Magazine*, 24(3):13–24.

[Gennari et al., 2003] Gennari, J. H., Musen, M. A., Fergerson, R. W., Grosso, W. E., Monica Crubézy, H. E., Noy, N. F., and Tu, S. W. (2003). The evolution of protégé: an environment for knowledge-based systems development. *International Journal of Human-Computer Studies*, 58(1):89–123.

[Goh, 1997] Goh, C. H. (1997). *Representing and reasoning about semantic conflicts in heterogeneous information systems*. PhD thesis, MIT Sloan School of Management.

[Grant et al., 1993] Grant, J., Litwin, W., Roussopoulos, N., and Sellis, T. (1993). Query languages for relational multidatabases. *The VLDB Journal*, 2(2):153–172.

[Gruber, 1993] Gruber, T. R. (1993). A translation approach to portable ontology specifications. *Knowl. Acquis.*, 5:199–220.

[Haav and Lubi, 2001] Haav, H.-M. and Lubi, T.-L. (2001). A Survey of Concept-based Information Retrieval Tools on the Web. In *Proceedings of the 5th East European Conferenceon Advances in Databases and Information Systems (ADBIS'01)*, pages 29–41.

[Halevy et al., 2004] Halevy, A. Y., Ives, Z. G., Madhavan, J., Mork, P., Suciu, D., and Tatarinov, I. (2004). The piazza peer data management system.

[Harris and Gibbins, 2003] Harris, S. and Gibbins, N. (2003). 3store: Efficient bulk rdf storage. In *Proceedings of the 1st International Workshop on Practical and Scalable Semantic Systems*.

[Harris, 1951] Harris, Z. (1951). *Methods in Structural Linguistics*. Chicago: University of Chicago Press.

[Horrocks et al., 2004] Horrocks, I., Patel-Schneider, P., Boley, H., Tabet, S., Grosof, B., and Dean, M. (2004). Swrl: A semantic web rule language combining owl and ruleml.

[Ian Niles, 2001] Ian Niles, A. P. (2001). Towards a standard upper ontology. In *Proceedings of the 2nd International Conference on Formal Ontology in Information Systems (FOIS'01)*, pages 2–9.

[IS010303.02, 1994] IS010303.02 (1994). Product data representation and exchange - part 2 : Express reference manual. Technical report, International Standards Organization, Genève.

[Isaac and Malaisé, 2003] Isaac, A. and Malaisé, V. (2003). Using xslt for interoperability: Doe and the travelling domain experiment. EON'03. International Workshop on Evaluation of Ontology-based Tools.

[ISO13584-25, 2004] ISO13584-25 (2004). Industrial automation systems and integration – Parts library – Part 25: Logical resource: Logical model of supplier library with aggregate values and explicit content. Technical report, International Standards Organization, Genève.

[ISO13584-42, 1998] ISO13584-42 (1998). Industrial automation systems and integration – Parts library – Part 42: Description methodology: Methodology for structuring parts families. Technical report, International Standards Organization, Genève.

[Itemis, 2011] Itemis (2011). *Xtext User Guide*. Itemis.

[Jean et al., 2006] Jean, S., Aït-Ameur, Y., and Pierra, G. (2006). Querying ontology based databases. The OntoQL proposal. In *Software Engineering and Knowledge Engineering*, pages 166–171.

[Jean et al., 2007] Jean, S., Aït-Ameur, Y., and Pierra, G. (2007). An Object-Oriented Based Algebra for Ontologies and their Instances. In *Proceedings of the 11th East European Conference in Advances in Databases and Information Systems (ADBIS'07)*, volume 4690 of *Lecture Notes in Computer Science*, pages 141–156. Springer.

[Jean et al., 2005] Jean, S., Pierra, G., and Ait-Ameur, Y. (2005). OntoQL: an exploitation language for OBDBs. In *Proceedings of the VLDB 2005 PhD Workshop. Co-located with the 31th International Conference on Very Large Data Bases (VLDB'05)*, pages 41–45.

[Jeff McAffer and Aniszczyk, 2010] Jeff McAffer, J.-M. L. and Aniszczyk, C. (2010). *Eclipse Rich Client Platform*. Addison Wesley.

[Jouault et al., 2008] Jouault, F., Allilaire, F., Bézivin, J., and Kurtev, I. (2008). Atl: a model transformation tool. In *Science of Computer Programming*.

[Jouault et al., 2006] Jouault, F., Bézivin, J., and Kurtev, I. (2006). Tcs:: a dsl for the specifica-

tion of textual concrete syntaxes in model engineering. In *Proceedings of the 5th international conference on Generative programming and component engineering*, GPCE'06, pages 249–254, New York, NY, USA. ACM.

[Kalnina et al., 2010] Kalnina, E., Kalnins, A., Celms, E., and Sostaks, A. (2010). Graphical template language for transformation synthesis. In *Proceedings of the Second international conference on Software Language Engineering*, SLE'09.

[Karvounarakis et al., 2002] Karvounarakis, G., Alexaki, S., Christophides, V., Plexousakis, D., and Scholl, M. (2002). RQL: a declarative query language for RDF. In *Proceedings of the Eleventh International World Wide Web Conference (WWW'02)*, pages 592–603.

[Karvounarakis et al., 2004] Karvounarakis, G., Magkanaraki, A., Alexaki, S., Christophides, V., Plexousakis, D., Scholl, M., and Tolle, K. (2004). RQL: A Functional Query Language for RDF. In Gray, P. M. D., Kerschberg, L., King, P. J. H., and Poulovassilis, A., editors, *The Functional Approach to Data Management: Modelling, Analyzing and Integrating Heterogeneous Data*, LNCS, pages 435–465. Springer-Verlag.

[Kelley et al., 1995] Kelley, W., Gala, S., Kim, W., Reyes, T., and Graham, B. (1995). Schema architecture of the unisql/m multidatabase system. In *Modern Database Systems*.

[Klampanos and Jose, 2003] Klampanos, I. A. and Jose, J. M. (2003). An architecture for peer-to-peer information retrieval. In *Proceedings of the 26th annual international ACM SIGIR conference on Research and development in informaion retrieval*, SIGIR '03, pages 401–402, New York, NY, USA. ACM.

[Kleppe et al., 2003] Kleppe, A. G., Warmer, J., and Bast, W. (2003). *MDA Explained: The Model Driven Architecture: Practice and Promise*. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA.

[Kolovos et al., 2008] Kolovos, D., Paige, R., and Polack, F. (2008). The epsilon transformation language. *Theory and Practice of Model Transformations*, 5063:46–60.

[Kurtev, 2008] Kurtev, I. (2008). Applications of graph transformations with industrial relevance. chapter State of the Art of QVT: A Model Transformation Language Standard, pages 377–393. Springer-Verlag, Berlin, Heidelberg.

[Lakshmanan et al., 1999] Lakshmanan, L. V. S., Sadri, F., and Subramanian, S. N. (1999). On Efficiently Implementing SchemaSQL on an SQL Database System. In Atkinson, M. P., Orlowska, M. E., Valduriez, P., Zdonik, S. B., and Brodie, M. L., editors, *Proceedings of 25th International Conference on Very Large Data Bases (VLDB'99)*, pages 471–482. Morgan Kaufmann.

[Lakshmanan et al., 2001] Lakshmanan, L. V. S., Sadri, F., and Subramanian, S. N. (2001). SchemaSQL: An Extension to SQL for Multidatabase Interoperability. *ACM Transactions on Database Systems (TODS)*, 26(4):476–519.

[Lefèvre, 2000] Lefèvre, P. (2000). *La Recherche d'informations. Du texte intégral au thésaurus*. Hermès-Lavoisier.

[Ling et al., 2001] Ling, Y., Miller, R. J., Haas, L. M., and Fagin, R. (2001). Data-driven understanding and refinement of schema mappings. In *SIGMOD Record*, pages 485–496.

[Litwin et al., 1989] Litwin, W., Abdellatif, A., Zeroual, A., Nicolas, B., and Vigier, P. (1989). MSQL: a Multidatabase Language. *Information Sciences: an International Journal*, 49(1-3):59–101.

[Ma et al., 2004] Ma, L., Su, Z., Pan, Y., Zhang, L., and Liu, T. (2004). Rstar: an rdf storage and query system for enterprise resource management. In *Proceedings of the 13th International Conference on Information and Knowledge*

Management, CIKM '04, pages 484–491, New York, NY, USA.

[Makris et al., 2010] Makris, K., Gioldasis, N., Bikakis, N., and Christodoulakis, S. (2010). Ontology mapping and sparql rewriting for querying federated rdf data sources. In *Proceedings of the 2010 international conference on On the move to meaningful internet systems: Part II*, OTM'10, pages 1108–1117, Berlin, Heidelberg. Springer-Verlag.

[Manola and Miller, 2004] Manola, F. and Miller, E. (2004). *RDF Primer*. World Wide Web Consortium. `http://www.w3.org/TR/rdf-primer`.

[Melnik et al., 2003a] Melnik, S., Rahm, E., and Bernstein, P. A. (2003a). Developing metadata-intensive applications with rondo. In *Journal of Semantic Web*.

[Melnik et al., 2003b] Melnik, S., Rahm, E., and Bernstein, P. A. (2003b). Rondo: a programming platform for generic model management. In *Proceedings of the 2003 ACM SIGMOD international conference on Management of data*, SIGMOD '03, pages 193–204, New York, NY, USA. ACM.

[Mens and Vangorp, 2006] Mens, T. and Vangorp, P. (2006). A taxonomy of model transformation. *Electronic Notes in Theoretical Computer Science*, 152(GraMoT):125–142.

[Miller and Mukerji, 2003] Miller, J. and Mukerji, J. (2003). Mda guide version 1.0.1. Technical report, Object Management Group (OMG).

[Minsky, 1975] Minsky, M. (1975). *A framework for representing knowledge*, pages 211–277. McGraw-Hill.

[Moha et al., 2010] Moha, N., Sen, S., Faucher, C., Barais, O., and Jézéquel, J.-M. (2010). Evaluation of kermeta for solving graph-based problems. In *Journal of Software tools for Technology Transfer*.

[Nash, 2005] Nash, A. (2005). Composition of mappings given by embedded dependencies. In *In PODS*, pages 172–183.

[Nathalie et al., 2008] Nathalie, A.-G., Sylvie, D., and Sylvie, S. (2008). The terminae method and platform for ontology engineering from texts. In *Proceedings of the 2008 conference on Ontology Learning and Population: Bridging the Gap between Text and Knowledge*, pages 199–223, Amsterdam, The Netherlands, The Netherlands. IOS Press.

[Noy, 2004] Noy, N. F. (2004). Semantic integration: A survey of ontology-based approaches. *SIGMOD Record*, 33(4):65–70.

[O'Conner, 2007] O'Conner, J. (2007). *Creating Extensible Applications With the Java Platform*. Sun Developper Network.

[OMG, 2003] OMG (2003). Uml 2.0 ocl specification. omg document final/03-10-14.

[Pan and Heflin, 2003] Pan, Z. and Heflin, J. (2003). Dldb: Extending relational databases to support semantic web queries. In *PSSS*.

[Paquette, 2005] Paquette, G. (2005). *Modélisation des connaissances et des compétences : un langage graphique pour concevoir et apprendre*. Sainte-Foy:Presses de l'Université du Québec.

[Park et al., 2007] Park, M.-J., Lee, J., Lee, C.-H., Lin, J., Serres, O., and Chung, C.-W. (2007). An efficient and scalable management of ontology. In *Proceedings of the 12th international conference on Database systems for advanced applications*, DASFAA'07, pages 975–980, Berlin, Heidelberg. Springer-Verlag.

[Patel-Schneider et al., 2004] Patel-Schneider, P. F., Hayes, P., and Horrocks, I. (2004). *OWL Web Ontology Language Semantics and Abstract Syntax*. World Wide Web Consortium. `http://www.w3.org/TR/owl-semantics/`.

[Patrascoiu, 2004] Patrascoiu, O. (2004). Yatl: Yet another transformation language - reference manual version 1.0. *Technical report University of Kent at Canterbury Computing Laboratory*.

[Paul Buitelaar, 2005] Paul Buitelaar, Philipp Cimiano, B. M. (2005). *Ontology Learning from Text: An Overview*, volume 123, chapter -. IOS Press.

[Petrini and Risch, 2007] Petrini, J. and Risch, T. (2007). Sward: Semantic web abridged relational databases. In *Proceedings of the 18th International Conference on Database and Expert Systems Applications*, pages 455–459.

[Petrov and Nemes, 2008] Petrov, I. and Nemes, G. (2008). A query language for mof repository systems. In *OTM '08*, pages 354–373.

[Pierra et al., 2005] Pierra, G., Dehainsala, H., Aït-Ameur, Y., and Bellatreche, L. (2005). Base de Données à Base Ontologique : principes et mise en œuvre. *Ingénierie des Systèmes d'Information*, 10(2):91–115.

[Rahm and Bernstein, 2001] Rahm, E. and Bernstein, P. A. (2001). A survey of approaches to automatic schema matching. *The VLDB Journal*, 10:334–350.

[Renée Miller, 2000] Renée Miller, Laura Haas, M. H. (2000). Schema mapping as query discovery. In *26th International Conference on Very Large Data Bases (VLDB'00)*, pages 77–88.

[Salton et al., 1982] Salton, G., Buckley, C., and Yu, C. T. (1982). An evaluation of term dependence models in information retrieval. In *Proceedings of the 5th annual ACM conference on Research and development in information retrieval*, SIGIR '82, pages 151–173, New York, USA. Springer-Verlag.

[Schenk and Wilson, 1994] Schenk, D. and Wilson, P. (1994). *Information Modelling The EXPRESS Way*. Oxford University Press.

[Seidewitz, 2003] Seidewitz, E. (2003). What models mean. *IEEE Softw.*, 20(5):26–32.

[Sirin et al., 2007] Sirin, E., Parsia, B., Grau, B., Kalyanpur, A., and Katz, Y. (2007). Pellet: A practical owl-dl reasoner. *Web Semantics Science Services and Agents on the World Wide Web*, 5(2):51–53.

[Smith et al., 2004] Smith, M. K., Welty, C., and McGuinness, D. L. (2004). *OWL Web Ontology Language Guide*. World Wide Web Consortium. `http://www.w3.org/TR/owl-guide/`.

[Staub and Maier, 1997] Staub, G. and Maier, M. (1997). *Ecco tool kit - an environnement for the evaluation of express models and the development of step based it applications.* User Manual.

[Steel and Lawley, 2004] Steel, J. and Lawley, M. (2004). Model-based test driven development of the tefkat model-transformation engine. In *Proceedings of the 15th International Symposium on Software Reliability Engineering*, ISSRE'04.

[Studer et al., 1998] Studer, R., Benjamins, V. R., and Fensel, D. (1998). Knowledge engineering: Principles and methods. *Data Knowl. Eng.*, 25(1-2):161–197.

[Sugumaran and Storey, 2006] Sugumaran, V. and Storey, V. C. (2006). The role of domain ontologies in database design: An ontology management and conceptual modeling environment. *ACM Transactions on Database Systems (TODS)*, 31(3):1064–1094.

[Sure et al., 2003] Sure, Y., Angele, J., and Staab, S. (2003). Ontoedit: Multifaceted inferencing for ontology engineering. *Journal of Data Semantics*, 1:128–152.

[Sure et al., 2002] Sure, Y., Staab, S., and Angele, J. (2002). Ontoedit: Guiding ontology development by methodology and inferencing. In *Proceedings of the International Conference on Ontologies,Databases and Applications of SEmantics ODBASE 2002*.

[Szulman et al., 2010] Szulman, S., Charlet, J., Aussenac-Gilles, N., Nazarenko, A., Hernandez, N., Nada, N., Sardet, E., Delahousse, J., and Téguiak, H. V. (2010). Dafoe : une plateforme multi-méthodes et multi-modèles pour construire des ontologies de domaine (demo). In *Congrès Francophone de Reconnaissance des Formes et Intelligence Artificielle*, RFIA'10, pages 1–2.

[Tetlow et al., 2005] Tetlow, P., Pan, J., Oberle, D., Wallace, E., Uschold, M., and Kendall, E. (2005). *Ontology Driven Architectures and Potential Uses of the Semantic Web in Systems and Software Engineering*. World Wide Web Consortium.

[Téguiak et al., 2010] Téguiak, H. V., Aït-Ameur, Y., Jean, S., and Sardet, E. (2010). Incremental design of ontologies: A model transformation-based approach. In *Proceedings of the International Conference on Knowledge Engineering and Ontology Development (KEOD'2010)*, pages 94–103.

[Téguiak et al., 2012a] Téguiak, H. V., Aït-Ameur, Y., Jean, S., and Sardet, E. (2012a). Mql: A mapping management language for model-based databases. In *Proceedings of the International Conference on Entreprise Information Systems (ICEIS'2012)*, pages 145–150.

[Téguiak et al., 2012b] Téguiak, H. V., Aït-Ameur, Y., and Sardet, E. (2012b). Use of persistent meta-modeling systems to handle mappings for ontology design. In *Proceedings of the International Conference on Models and Ontology-based Design of Protocols, Architectures and Services (MOPAS'2012)*, pages –.

[Theoharis et al., 2005] Theoharis, Y., Christophides, V., and Karvounarakis, G. (2005). Benchmarking database representations of rdf/s stores. In *International Semantic Web Conference*, pages 685–701.

[Tratt, 2005] Tratt, L. (2005). Model transformations and tool integration. *Journal of Software and Systems Modelling*, 4(2):112–122.

[Volker Haarslev, 2001] Volker Haarslev, R. M. (2001). Racer system description. In *Proceedings of the First International Joint Conference on Automated Reasoning*, IJCAR '01, pages 701–706, London, UK, UK. Springer-Verlag.

[Wache et al., 2001] Wache, H., Vögele, T., Visser, U., Stuckenschmidt, H., Schuster, G., Neumann, H., and Hübner, S. (2001). Ontology-based integration of information — a survey of existing approaches. In *Proceedings of the IJCAI-01 Workshop: Ontologies and Information Sharing*, pages 108–117.

[Wakeman and Jowett, 1993] Wakeman, L. and Jowett, J. (1993). *PCTE: the standard for open repositories*. Prentice-Hall, Inc., Upper Saddle River, NJ, USA.

[Wilkinson et al., 2003] Wilkinson, K., Sayers, C., Kuno, H., and Reynolds, D. (2003). Efficient RDF storage and retrieval in Jena2. In *Proc. First International Workshop on Semantic Web and Databases*.

[Wüster, 1981] Wüster, E. (1981). *L'étude scientifique générale de la terminologie, zone frontalière entre la linguistique, la logique, l'ontologie, l'informatique et les sciences des choses. Textes choisis de terminologie 1, Fondements théoriques de la terminologie*. Presses de l'université de Laval.

[Zadeh, 1965] Zadeh, L. A. (1965). Fuzzy sets. *Information and Control*, 8(3):338–353.

Publications

1. Henry Valéry Téguiak, Yamine Aït-Ameur, Stéphane Jean, Eric Sardet, MQL: A Mapping Management Language for Model-Based Databases, in Proceedings of the International Conference on Entreprise Information Systems (ICEIS'2012), pages 145-150, Wroclaw, Poland, June, 2012.

2. Henry Valéry Téguiak, Yamine Aït-Ameur, Stéphane Jean, Eric Sardet, Use of persistent metamodeling systems to handle mappings for ontology design, in Proceedings of the International Conference on Models and Ontology-based Design of Protocols, Architectures and Services (MOPAS'2012), pages 145-150, Chamonix, France, May, 2012.

3. Henry Valéry Téguiak, Yamine Aït-Ameur, Stéphane Jean, Eric Sardet, Incremental design of ontologies: A model transformation-based approach, in Proceedings of the International Conference on Knowledge Engineering and Ontology Development (KEOD'2010), pages 94-103, Valencia, Spain, October, 2010.

4. Jean Charlet, Sylvie Szulman, Nathalie Aussenac-Gilles, Adeline Nazarenko, Henry Valéry Téguiak, Eric, Sardet, DaFOE: A Platform for Building Ontologies from Texts (Poster and demo), in Proceedings of the International Conference on Knowledge Engineering and Knowledge Management (EKAW'2010), Lisbonne, Portugal, October, 2010.

5. Jean Charlet, Sylvie Szulman, Nathalie Aussenac-Gilles, Adeline Nazarenko, Nathalie Hernandez, Nadiah Nada, Eric Sardet, Jean Delahousse, Henry Valéry Téguiak, Audrey Baneyx, DAFOE : une plate-forme pour construire des ontologies à partir de textes et de thésaurus (poster), Actes du Congrès Francophone de Reconnaissance des Formes et Intelligence Artificielle (RFIA'2010), pages 1-3, Hammamet, Tunisie, Janvier, 2010.

6. Sylvie Szulman, Jean Charlet, Nathalie Aussenac-Gilles, Adeline Nazarenko, Nathalie Hernandez, Nadia Nada, Eric Sardet, Jean Delahousse, Henry Valéry Téguiak, DAFOE : une plate-forme multi-méthodes et multi-modèles pour construire des ontologies de domaine (demo), Actes des Journées Francophones Extraction et Gestion de Connaissances (EGC'2010), pages 1-2, Caen,

France, Janvier, 2010.

7. Henry Valéry Téguiak, Construction d'ontologies à partir de textes : une approche basée sur l-Ingénierie Dirigée par les Modèles, XXVIIème Congrès INFORSID (INFORSID'09), Toulouse, France, Mai, 2009.

8. Jean Charlet, Sylvie Szulman, Guy Pierra, Nadia Nadiah, Henry Valéry Téguiak, Nathalie Aussenac-Gilles, Adeline Nazarenko, DaFOE: A multimodel and multimethod platform for building domain ontologies, Actes des 2ièmes Journées Francophones sur les Ontologies (JFO'2008), Lyon, France, 2008.

Annexe A

Structure du langage MQL

Cette annexe présente la grammaire du langage MQL. Nous utilisons les symboles suivants pour description de cette grammaire:

- | = alternative (OU)
- [] = optionnel
- * = répétition (zéro ou plusieurs)
- {} = obligatoire

- Root

<mql_statement> ::= <select_statement> | <update_statement> | <delete_statement> | <insert_statement>

- Select

<select_statement> ::= <select_clause> <from_clause> [<where_clause>] [<groupby_clause>] [<having_clause>] [<orderby_clause>] [<match_clause>] [<filter_clause>] [<confidence_clause>] [<with_closure_clause>] [depth_clause] [mwhere_clause]

<select_clause> ::= **SELECT** [**DISTINCT**] <select_expression> {, <select_expression>}*

<select_expression> ::= <aggregate_expression> | <identification_variable>

<aggregate_expression> ::= { **AVG** | **MAX** | **MIN** | **SUM** } ([**DISTINCT**] <state_field_path_expression>) | **COUNT** ([**DISTINCT**] <identification_variable> | <state_field_path_expression>)

<state_field_path_expression> ::= [<identification_variable>.]<state_field>

<identification_variable_declaration> ::= <range_variable_declaration>

<range_variable_declaration> ::= <abstract_schema_name> [**AS**] <identification_variable>

- From

<from_clause> ::= **FROM** <identification_variable_declaration> {, <identification_variable_declaration>}*

- Where

<where_clause> ::= **WHERE** <conditional_expression>

<conditional_expression> ::= <conditional_term> | <conditional_expression> **OR** <conditional_term>

<conditional_term> ::= <conditional_factor> | <conditional_term> **AND** <conditional_factor>

<conditional_factor> ::= [**NOT**] <conditional_primary>

<conditional_primary> ::= <simple_cond_expression> | (<conditional_expression>)

<simple_cond_expression> ::= <comparison_expression> | <between_expression> | <like_expression> | <in_expression> | <null_comparison_expression> | <exists_expression>

<between_expression> ::= <arithmetic_expression> [**NOT**] **BETWEEN** <arithmetic_expression> **AND** <arithmetic_expression> | <string_expression> [**NOT**] **BETWEEN** <string_expression> **AND** <string_expression> | <datetime_expression> [**NOT**] **BETWEEN** <datetime_expression> **AND** <datetime_expression>

<in_expression> ::= <state_field_path_expression> [**NOT**] **IN** (<in_item> {, <in_item>}* | <subquery>)

<in_item> ::= <literal> | <input_parameter>

<like_expression> ::= <string_expression> [**NOT**] **LIKE** <pattern_value> [ESCAPE <escape_character>]

<null_comparison_expression> ::= <input_parameter> **IS** [**NOT**] **NULL**

<exists_expression> ::= [**NOT**] **EXISTS** (<subquery>)

<comparison_expression> ::= <string_expression comparison_operator> <string_expression> | <boolean_expression> { =|<>} <boolean_expression> | <enum_expression> { =|<>} <enum_expression> | <datetime_expression> <comparison_operator> <datetime_expression> | <entity_expression> { = | <> } <entity_expression> | <arithmetic_expression> <comparison_operator> <arithmetic_expression>

<comparison_operator> ::= = | > | >= | < | <= | <>

<arithmetic_expression> ::= <simple_arithmetic_expression> | (<subquery>)

<simple_arithmetic_expression> ::= <arithmetic_term> | <simple_arithmetic_expression> { + | - } <arithmetic_term>

<arithmetic_term> ::= <arithmetic_factor> | <arithmetic_term> { * | / } <arithmetic_factor>

<arithmetic_factor> ::= [{ + | - }] <arithmetic_primary>

<arithmetic_primary> ::= <numeric_literal> | (<simple_arithmetic_expression>) | <input_parameter> | <functions_returning_numerics> | <aggregate_expression>

<string_expression> ::= <string_primary> | (<subquery>)

<string_primary> ::= <string_literal> | <input_parameter> | <functions_returning_strings> | <aggregate_expression>

<datetime_expression> ::= <datetime_primary> | (<subquery>)

<datetime_primary> ::= <input_parameter> | <functions_returning_datetime> | <aggregate_expression>

<boolean_expression> ::= <boolean_primary> | (<subquery>)

<boolean_primary> ::= <boolean_literal> | <input_parameter>

<enum_expression> ::= <enum_primary> | (<subquery>)

<enum_primary> ::= <enum_literal> | <input_parameter>

<entity_expression> ::= <single_valued_association_path_expression> | <simple_entity_expression>

<simple_entity_expression> ::= <identification_variable> | <input_parameter>

- Functions

<functions_returning_numerics> ::= LENGTH(<string_primary>) | LOCATE(<string_primary>, <string_primary>[, <simple_arithmetic_expression>]) | ABS(<simple_arithmetic_expression>) | SQRT(<simple_arithmetic_expression>)

<functions_returning_datetime> ::= CURRENT_DATE | CURRENT_TIME | CURRENT_TIMESTAMP

<functions_returning_strings> ::= CONCAT(<string_primary>, <string_primary>) | SUBSTRING(<string_primary>, <simple_arithmetic_expression>, <simple_arithmetic_expression>)

- Group By

<groupby_clause> ::= **GROUP BY** <groupby_item> {, <groupby_item>}*

<groupby_item> ::= <identification_variable>

<having_clause> ::= HAVING <conditional_expression>

- Order By

<orderby_clause> ::= **ORDER BY** <orderby_item> {, <orderby_item>}*

<orderby_item> ::= <state_field_path_expression> [ASC | DESC]

- Match

<match_clause> ::= **MATCH** <match_items>

<match_items> ::= ([<identification_variable>.]<state_field> {, [<identification_variable>.]<state_field>}*)

- Filter

<filter_clause> ::= **FILTER** <filter_items>

<filter_items> ::= ([<identification_variable>.]<state_field> {, [<identification_variable>.]<state_field>}*)

- Confidence

<confidence_clause> ::= **CONFIDENCE** <normalized_integer>

- With Closure

<with_closure_clause> ::= **With Closure**

- Depth

<depth_clause> ::= Integer

- mWhere

<mwhere_clause> ::= **mWHERE** <conditional_expression>

- Subquery

<subquery> ::= <simple_select_clause> <subquery_from_clause> [<where_clause>] [<groupby_clause>] [<having_clause>]

<subquery_from_clause> ::= **FROM** <subselect_identification_variable_declaration> {, <subselect_identification_variable_declaration>}*

<subselect_identification_variable_declaration> ::= <identification_variable_declaration> | <association_path_expression> [**AS**] <identification_variable> | <collection_member_declaration>

<simple_select_clause> ::= **SELECT** [**DISTINCT**] <simple_select_expression>

<simple_select_expression> ::= <aggregate_expression> | <identification_variable>

- Update

<update_statement> ::= <update_clause> [<where_clause>] [<match_clause>] [<filter_clause>] [<confidence_clause>] [<with_closure_clause>] [<depth_clause>] [<mwhere_clause>]

<update_clause> ::= **UPDATE** <abstract_schema_name> [[AS] <identification_variable>] **SET** <update_item> {, <update_item>}*

<update_item> ::= [<identification_variable>.]<state_field> = <new_value>

<new_value> ::= <simple_arithmetic_expression> | <string_primary> | <datetime_primary> | <boolean_primary> | <enum_primary> | <simple_entity_expression> | null

- Delete

<delete_statement> ::= <delete_clause> [<where_clause>] [<match_clause>] [<filter_clause>] [<confidence_clause>] [<with_closure_clause>] [<depth_clause>] [<mwhere_clause>]

<delete_clause> ::= **DELETE FROM** <abstract_schema_name> [[**AS**] <identification_variable>]

- Insert

<insert_statement> ::= **INSERT INTO** <insert_clause> [<match_clause>] [<filter_clause>] [<confidence_clause>] [<with_closure_clause>] [<depth_clause>] [<mwhere_clause>]

<insert_clause> ::= <abstract_schema_name> [<insert_items>] **VALUES** <new_values>

<insert_items> ::= ([<identification_variable>.]<state_field> {, [<identification_variable>.]<state_field>}*)

<new_values> ::= (<new_value> {, <new_value>}*)

- Create

<create_statement> ::= **STRUCT** <create_clause>

<create_clause> ::= <abstract_schema_name> (<create_item> {, <create_item>}*)

<create_item> ::= <state_field> <literal> | <state_field> **REF** (<abstract_schema_name>)

- Literals

literal ::= String | Integer | Long | Float | Double | Boolean | Date | Time | TimeStamp | null | normalized_integer String - 'string'

Integer - digits

Long - +|-digitsL

Float - +|-digits.decimaleexponentF

Double - +|-digits.decimaleexponentD

Boolean - TRUE | FALSE

Date - {d'yyyy-mm-dd'}

Time - {t'hh:mm:ss'}

Timestamp - {ts'yyy-mm-dd hh:mm:ss.nnnnnnnnn'}

null - NULL

Le littéral `normalized_integer` est un entier compris entre 0 et 1.

Table des figures

Liste des tableaux

Glossaire

API : Application Programming Interface

BDBM : Base de Données à Base de Modèles

BDBO : Base de Données à Base Ontologique

DaFOE : Differential and Formal Ontologies Editor

DaFOE4App : Differential and Formal Ontologies Editor for Application

DML : Data Manipulation Language

DQL : Data Query Language

EBNF : Extended Backus-Naur Form

IDM : Ingénierie Dirigée par les Modèles

MOF : Meta Object Facility

MOT : Modélisation par Objets Typés

MQL : Mapping Query Language

OMG : Object Management Group

OWL : Web Ontology Language

PLIB : Parts Library - Norme ISO 13584

RDF : Ressource Description Framework

SGBD : Système de Gestion de Base de Données

SQL : Strutured Query Language

TF-IDF : Term Frequency - Inverse Document Frequency

UML : Unified Modeling Language

Construction d'ontologies à partir de textes : une approche basée sur les transformations de modèles

Henry Valéry TEGUIAK

Résumé. Depuis son émergence au début des années 1990, la notion d'ontologie s'est rapidement diffusée dans un grand nombre de domaines de recherche. Compte tenu du caractère prometteur de cette notion, de nombreux travaux portent sur l'utilisation des ontologies dans des domaines aussi divers que la recherche d'information, le commerce électronique, le web sémantique, l'intégration de données, etc. L'efficacité de tous ces travaux présuppose l'existence d'une ontologie de domaine susceptible d'être utilisée. Or, la conception d'une telle ontologie s'avère particulièrement difficile si l'on souhaite qu'elle fasse l'objet de consensus. S'il existe des outils utilisés pour éditer une ontologie supposée déjà conçue, et s'il existe également plusieurs plate-formes de traitement automatique de la langue permettant d'analyser automatiquement les corpus et de les annoter tant du point de vue syntaxique que statistique, il est difficile de trouver une procédure globalement acceptée, ni a fortiori un ensemble d'outils supports permettant de concevoir une ontologie de domaine de façon progressive, explicite et traçable à partir d'un ensemble de ressources informationnelles relevant de ce domaine. L'objectif du projet ANR DaFOE4App (Differential and Formal Ontologies Editor for Application), dans lequel s'inscrit notre travail, était de favoriser l'émergence d'un tel ensemble d'outils. Contrairement à d'autres outils de construction d'ontologies, la plate-forme DaFOE, présentée dans cette thèse, ne propose pas un processus de construction figé ni en nombre d'étapes, ni sur la représentation des étapes. En effet, dans cette thèse nous généralisons le processus de construction d'ontologies pour un nombre quelconque d'étapes. L'intérêt d'une telle généralisation étant, par exemple, d'offrir la possibilité de raffiner le processus de construction en insérant ou modifiant des étapes. On peut également souhaiter supprimer certaines étapes à fin de simplifier le processus de construction. L'objectif de cette généralisation est de minimiser l'impact de l'ajout, suppression ou modification d'une étape dans le processus global de construction d'ontologies, tout en préservant la cohérence globale du processus de construction. Pour y parvenir, notre approche consiste à utiliser l'Ingénierie Dirigée par les Modèles pour caractériser chaque étape au sein d'un modèle et ensuite ramener le problème du passage d'une étape à l'autre à un problème de mapping de modèles. Les mappings établis entre les modèles sont ensuite utilisés pour semi-automatiser le processus de construction d'ontologies. Ce processus de construction se faisant dans un contexte persistant de base de données, nous proposons dans cette thèse, d'une part, pour les bases de données dites à base de modèles (BDBM) du fait qu'elles permettent de stocker à la fois les données et les modèles décrivant ces données, une extension pour la prise en compte des mappings, et, d'autre part, nous proposons le langage de requête MQL (Mapping Query Language) qui, en masquant la complexité de l'architecture de la BDBM facilite son exploitation. L'originalité du langage MQL se trouve dans sa capacité, au travers de requêtes syntaxiquement compactes, à explorer transitivement tout ou partie du graphe de mappings lors d'une recherche d'informations.

Mots-clés : Base de données à base de modèles, Langage de requête, Mappings, Méta-modélisation, Ontologie, Transformation de modèles.

.....

Abstract. Since its emergence in the early 1990s, the notion of ontology has been quickly distributed in many areas of research. Given the promise of this concept, many studies focus on the use of ontologies in many areas like information retrieval, electronic commerce, semantic Web, data integration, etc.. The effectiveness of all this work is based on the assumption of the existence of a domain ontology that is already built an that can be used. However, the design of such ontology is particularly difficult if you want it to be built in a consensual way. If there are tools for editing ontologies that are supposed to be already designed, and if there are also several platforms for natural language processing able to automatically analyze corpus of texts and annotate them syntactically and statistically, it is difficult to find a globally accepted procedure useful to develop a domain ontology in a progressive, explicit and traceable manner using a set of information resources within this area. The goal of ANR DaFOE4App (Differential and Formal Ontology Editor for Application) project, within which our work belongs to, was to promote the emergence of such a set of tools. Unlike other tools for ontologies development, the platform DaFOE presented in this thesis does not propose a methodology based on a fixed number of steps with a fixed representation of theses steps. Indeed, in this thesis we generalize the process of ontologies development for any number of steps. The interest of such a generalization is, for example, to offer the possibility to refine the development process by inserting or modifying steps. We may also wish to remove some steps in order to simplify the development process. The aim of this generalization is for instance, for the overall process of ontologies development, to minimize the impact of adding, deleting, or modifying a step while maintaining the overall consistency of the development process. To achieve this, our approach is to use Model Driven Engineering to characterize each step through a model and then reduce the problem of switching from one step to another to a problem of models transformation. Established mappings between models are then used to semi-automate the process of ontologies development. As all this process is stored in a database, we propose in this thesis, for Model Based Database (MBDB) because they can store both data and models describing these data, an extension for handling mappings. We also propose the query language named MQL (Mapping Query Language) in order to hide the complexity of the MBDB structure. The originality of the MQL language lies in its ability, through queries syntactically compact, to explore the graph of mappings using the transitivity property of mappings when retrieving informations.

Key Words : Mappings, Metamodelling, Model-based Database, Model transformation, Ontology, Query language.