Laboratoire d'InfoRmatique en Images et Systèmes d'information
LIRIS FRE2672 CNRS, INSA Lyon, UCB Lyon 1, EC Lyon

Ecole Doctorale
« Informatique et Information pour la Société »
EDA 335

Méthodes et outils Pour l'intégration des ontologies

Mémoire de stage de DEA
« Documents Multimédia, Images, et Systèmes d'Information Communicants »

Samer Abdul Ghafour **2003-2004**

Résumé

Notre stage se situe dans le cadre du « Web Sémantique », et de la représentation des connaissances, où l'ontologie joue un rôle essentiel. La conception coopérative entre les différents acteurs du monde industriel a créé le besoin d'échange de données et de la communication. Dans ce cadre, les représentations de leurs domaines sont souvent réalisées sous formes d'ontologies. Dans un même domaine, plusieurs projets opérationnels et de recherches ont abouti à la création d'autant d'ontologies, qui divergent tant au niveau du format que de la structure de données.
Notre stage vise à faciliter la définition des règles de mise en correspondance entre les concepts de différentes ontologies et les relations entre eux. Nous proposons une approche qui consiste à établir un mécanisme de conversion des différents formats vers le langage OWL. Nous avons appliqué notre approche dans le domaine de la construction et des bâtiments sur trois ressources : bcXML, e-Cognos, et Edibatec. Nous avons utilisé le langage de modélisation UML pour représenter les modèles spécifiques. Les langages de représentations utilisés sont RDF, RDFS, DAML+OIL, et les sous-langages de OWL.

Mots-clés: ontologie, Web sémantique, règles de correspondances, hétérogénéité, intégration d'ontologies, contraintes.

Abstract

This work is in tight relation to the concept of "Semantic Web", and the field of Knowledge Representation. "Ontology" is a basic term in the context of our work. Cooperation between different factors in the industrial domain has created the need to improve the data exchange and communication. Within this framework, the Knowledge Representation of their data is often carried out in forms of ontology. Within a domain, several operational projects and research has led to the existence of many ontologies to represent their data. These ontologies are dissimilar in their format as well as in the structure of data. We aim in our work to facilitate the mechanism of defining rules of mapping between the concepts of different ontologies, and the relations between them. To accomplish our goal, we propose an approach which consists in establishing a mechanism of conversion of the various formats towards the ontology language OWL. We applied our approach in the field of "Building and Construction" to three resources: bcXML, e-Cognos, and Edibatec. We have used the modeling language UML to represent the specific models. The languages of representations used are RDF, RDFS, DAML+OIL, and the sub-languages of OWL: Lite, DL, and Full.

Keywords: ontology, Semantic Web, mapping rules, heterogeneity, language expressivity, ontology integration, constraints.

REMERCIEMENTS

J'adresse tout d'abord mes vifs remerciements à M. Lionel MEDINI, ainsi qu'à Madame Parisa GHOUDOUS de m'avoir encadré pendant ce stage. Mes remerciements s'adressent aussi aux membres de jury, et les rapporteurs qui ont accepté d'évaluer mon travail.

Je n'oublie pas les personnes avec lesquelles j'ai partagé le souci du bon accomplissement du travail, notamment Catarina Ferreira Da Silva pour son soutien, et messieurs Celson LIMA et Bruno FIES du CSTB qui n'ont jamais hésité à répondre à toutes nos questions et nous fournir tous documents nécessaires. Merci également à Patrick Hoffmann ; sa sympathie a toujours créé une bonne ambiance de travail.

TABLES DE MATIERES

INTRODUCTION

Le cycle de développement de produit dans l'industrie est passé à l'ère de la conception coopérative. De nombreux experts, dans des domaines similaires ou différents, échangent des données et des documents pour parvenir à un résultat commun. Pour les entreprises confrontées à l'intensification de la concurrence et à l'accélération du changement, le partage d'informations et de connaissances apparaît de plus en plus comme un outil nécessaire pour améliorer l'efficacité de l'organisation et stimuler capacité d'innovation.

Le volume des données échangées devenant de plus en plus important, les problèmes communicationnels se posent de façon particulièrement aiguë au sein de l'entreprise. Ces problèmes peuvent provenir de différents facteurs. L'un d'eux est d'ordre syntaxique et réside dans l'hétérogénéité des formats de données, de la terminologie, des méthodologies ou encore des logiciels utilisés pour la conception. Cette hétérogénéité rend difficile l'échange de données entre les différents acteurs de l'entreprise.

Par ailleurs, pour rendre efficace l'échange de données, il est nécessaire de prendre en compte la sémantique qui leur est attachée. Cependant, dans les approches de traitement classiques, cette sémantique est cachée dans les composants des logiciels qui les traitent [1], ce qui crée des problèmes d'interopérabilité lors de l'échange des données. De nombreux travaux de recherche dans les domaines de la modélisation et de la représentation des connaissances ont été menés pour rendre explicite cette sémantique, d'une manière déclarative, avec le minimum d'ambiguïtés possible [2].

Différentes approches ont été développées pour représenter les connaissances spécifiques aux processus de conception. Les premières étaient basées sur les mots clés. Elles forment un moyen rapide pour retrouver des informations dans une structure simple, mais la complexité de l'information a rapidement dépassé ces approches. Une deuxième approche [3] est basée sur les dictionnaires qui permettent la classification des termes dans des catégories plus spécifiques et plus appropriées. Une troisième approche se base sur les taxonomies : elle ajoute la puissance de l'héritage qui permet de transmettre des significations généralisées vers les données spécialisées. Enfin, la dernière approche pour la représentation de la connaissance est basée sur les ontologies [4]. Les ontologies fournissent une structuration des données plus complexe que les taxonomies. Elles permettent une plus grande variété de relations structurelles et non structurelles pour représenter concrètement les concepts existants dans un domaine et les différentes relations entre eux.

De nombreux travaux de recherche et des projets opérationnels ont abouti à la création de multiples ontologies, parfois pour un même domaine. Dès lors, les problèmes d'interopérabilité se posent à nouveau, mais cette fois au niveau des formalismes de représentation des connaissances : ces ontologies sont souvent hétérogènes, à la fois au niveau de leur structure et au niveau de la sémantique des données qu'elles contiennent.

Contexte du travail

Ce stage s'effectue dans le cadre de l'Équipe « Modélisation Coopérative », de l'axe « Modélisation et réalité augmentée » du laboratoire LIRIS[1]. Les axes de recherche de cette équipe sont l'échange et le partage des données et de documents, la réalité augmentée et l'ingénierie des connaissances. Ces trois axes sont orientés dans une perspective de modélisation des processus de travail coopératif. Ce stage est co-encadré par Lionel Médini et Parisa Ghodous. Ce stage se déroule en collaboration avec un autre stage de DEA, réalisé par Patrick Hoffmann, et dont le sujet concerne la contextualisation des règles de correspondance entre les concepts des ontologies.

L'application choisie pour ce stage se déroule dans le domaine du bâtiment et de la construction (BC), en collaboration avec le CSTB[2]. L'industrie BC s'efforce perpétuellement d'améliorer les méthodes et les outils mis à disposition des professionnels du domaine. Pour cela, différents projets ont été mis en œuvre, axés sur les normes et standards, sur la structuration des échanges de données entre les différentes applications, puis sur la sémantique de ces données. Dans ce stage, nous nous appuyons sur les données générées par trois de ces projets opérationnels.

- *eConstruct* est un projet de recherche européen (IST) qui a permis de développer un standard de structuration des données techniques, appelé « *Building & Construction eXtensible Mark-up Language (bcXML)* ». Ce standard est largement utilisé dans le domaine de la construction, et a permis de définir une large collection de classifications des données techniques.

- *e-Cognos* est également un projet européen, et a permis de spécifier et de développer une infrastructure ouverte orientée modèles et un ensemble d'outils permettant une gestion cohérente de la connaissance dans les environnements collaboratifs de construction. L'analyse de la sémantique a conduit au développement d'ontologies et de mécanismes adaptifs qui permettent d'organiser des documents par rapport à leur contenu et leurs interdépendances.

- *Edibatec* est un projet mené par une association de partenaires de différents sous-domaines de la construction (génie climatique, plomberie…) ou connexes (édition de logiciels) ainsi que des organisations professionnelles. Cette association a permis aux partenaires de s'entendre sur un vocabulaire commun du domaine, afin de réaliser et de mettre en ligne un catalogue électronique des produits de construction.

Les ressources terminologiques et sémantiques issues de ces trois projets sont mises en commun dans le serveur d'ontologie « eCoser » (e-Cognos Ontology Server), actuellement développé au CSTB, et qui vise à permettre l'accès aux services Web aux acteurs de la construction, en tirant partie de l'intégration des multiples ressources sémantiques du domaine. Le but de l'application de ce stage est de mettre en relation ces ressources au sein du serveur pour améliorer l'offre de services.

[1] Laboratoire d'InfoRmatique en Image et Systèmes d'information : http://liris.cnrs.fr
[2] Centre Scientifique et Technique du Bâtiment : http://www.cstb.fr

Problématique

La diversité des formats de représentation de la connaissance pour un même domaine conduit à l'apparition des problèmes d'hétérogénéité entre différentes ontologies. Cette hétérogénéité se situe au niveau de la sémantique, et également au niveau de la syntaxe. La sémantique est liée à l'étude du contexte dans lequel les données sont représentées, et elle est identifiée par une étude sur le contenu des données. La syntaxe implique la structure du formalisme de représentation de la connaissance, et elle varie en fonction du langage choisi. La hétérogénéité s'aperçoit clairement durant les processus d'intégration des différentes ontologies.

Nous visons dans notre travail à faciliter le mécanisme de la mise en correspondance entre plusieurs concepts et relations présentés dans différentes ontologies. Nous nous limitions au traitement de la hétérogénéité syntaxique, en définissant des moyens pour représenter ces différentes ontologies dans un seul langage, le standard OWL. Ceci permet de faciliter leur compréhension et leur communication. D'où, il est indispensable de comprendre la sémantique au niveau des formalismes de représentations afin d'étudier l'expressivité des langages. Cette conversion homogénéise le format de représentation et par conséquent rend possible la mise en correspondance désirée.

Nous travail débute dans le chapitre I par une étude bibliographie où nous abordons par une définition de la sémantique, et les différents formalismes de représentation de la connaissance. Après, nous déplaçons, vers la définition de différents langages de représentation de la connaissance. Ensuite, nous nous intéressons aux différentes approches existantes pour l'intégration des ontologies, et les problèmes d'hétérogénéité qui y sont rattachés.

Le deuxième chapitre comprend notre approche méthodologique, où nous essayerons de détailler la démarche suivie, et la description du travail pratique réalisé. Enfin la conclusion résume les apports, et nos perspectives sur le travail, et une prévision sur les travaux futurs.

CHAPITRE I

ETAT DE L'ART

« The Semantic Web is an extension of the current web in which information is given well-defined meaning, better enabling computers and people to work in cooperation . »
Tim Berners-Lee

Le Web Sémantique fournit un cadre de travail commun qui permet à des données d'être partagées et réutilisées à travers l'application, l'entreprise, et les frontières de la communauté. C'est un effort de collaboration mené par le W3C[3] avec la participation d'un grand nombre de chercheurs et d'industriels associés. Il est fondé sur RDF « *Resource Description Framework* », qui intègre une variété d'applications en utilisant XML pour la syntaxe et l'URIs pour l'identification des ressources [5]. Ces opérations sont d'autant plus simples que les ressources sont identifiées de manières uniques et organisées autour d'une d'ontologie.

Concrètement, le Web sémantique peut être vu comme une infrastructure au sein de laquelle chaque ressource est couplée à des méta-données définies à l'aide de langages qui permettent de décrire les ressources ainsi que les relations qui les lient à l'aide de marqueurs sémantiques. Dans ce domaine, deux langages sont aujourd'hui en concurrence [6] : d'un côté le couple RDF et OWL, supportés par le W3C, et de l'autre Topic Maps [7], un standard proposé par l'ISO. Ces langages constituent un socle pour définir des ontologies, entités servant à faciliter l'exploitation automatique ou semi-automatique du contenu par un ordinateur tout en gardant la signification du contenu pour les êtres humains.

L'abondance des données sur le Web entrave la construction d'un agent Web capable de répondre aux requêtes des utilisateurs. D'où la nécessité d'améliorer la recherche par 'mots clés' en définissant une sémantique associée aux différentes ressources du Web [8].

Dans ce chapitre, nous présentons, dans un premier temps, une introduction permettant de définir la sémantique et son rôle dans la notion de « Web Sémantique ». Ensuite nous présentons les différents formalismes de représentation de la connaissance : les taxonomies, les différents formalismes pour construire des ontologies, notamment KIF, la logique de description, et langages DAML + OIL et OWL, avec les différences qui existent entre ces deux derniers.

Puis, nous nous déplaçons vers le modèle orienté objet, en détaillant le langage de modélisation UML, et sa combinaison avec un langage de définition de contraintes, OCL, pour la définition d'ontologies. Cela nous amène à nous intéresser aux problèmes d'hétérogénéité entre les ontologies lors de l'intégration des ontologies. Enfin, nous concluons avec une synthèse sur ce que nous avons vu, et qui nous permet de spécifier notre approche et d'aborder la partie pratique de notre travail.

[3] www.w3.org

I.1. Les techniques de Représentation de la connaissance

Dans œtte partie, nous distinguons les différents formalismes de représentation de la connaissance : les taxonomies, les ontologies, et les autres formalismes pour représenter la connaissance, avec les langages qui s'appuient sur ces types de formalismes.

I.1.1. Les Taxonomies

I.1.1.1. Définition

La ***taxonomie***, ou ***taxinomie*** (du grec taxis : rangement et nomos : loi) est **l'étude théorique de la classification**, de ses bases, de ses principes, des méthodes et des règles. La taxonomie permet d'organiser un vocabulaire sous une forme hiérarchique simple. Cette hiérarchisation correspond souvent à une spécialisation, autrement dit, c'est la classification des termes d'un domaine dans une hiérarchie.

I.1.1.2. Langages

XML est un langage puissant pour décrire une taxonomie. L'énorme évolution de la technologie a imposé l'écriture des données d'une manière qui permet de faciliter son échange. XML (*Extensible Markup Language*) est fourni comme un moyen pour accomplir cette tâche, en utilisant un langage de marquage basé sur le texte pour décrire la structure des données ordonnées [11]. Les fichiers XML sont liés à des définitions des types de documents (DTDs) qui décrivent la structure des documents XML. Mais les DTDs manquent d'expressivité pour décrire des structures de données de haut niveau [12]. XML Schéma [13] est un ensemble plus riche en structures, en types, et en contraintes pour décrire des données. Il propose des mécanismes de haut niveau pour la définition et la structuration de documents XML [12].

I.1.2. Les Ontologies

I.1.2.1. Définition

« An explicit specification of a conceptualisation » Tom Gruber

« Ontologie » est un terme emprunté à la philosophie qui implique une branche de la philosophie qui traite la nature et l'organisation de la réalité. Plusieurs définitions d'ontologies sont données dans la dernière décade, mais celle qui caractérise l'essentiel d'une ontologie est fondée sur la définition reliée dans [9] : « Une ontologie est une **spécification formelle**, et **explicite** d'une c**onceptualisation partagée** ». *Conceptualisation* réfère à un modèle abstrait de certains phénomènes dans le monde qui identifie les concepts appropriés de ce phénomène. *Explicite* signifie que le type de concepts utilisés et les contraintes sur leur utilisation doivent être explicitement définis. *Formelle* réfère au fait qu'une ontologie doit être compréhensible par la machine, c'est-à-dire cette dernière soit capable d'interpréter la sémantique de l'information fournie. *Partagée* indique que l'ontologie supporte la connaissance consensuelle, et elle n'est pas restreinte à certains individus mais accepté par un groupe [10].

Les taxonomies ou les thésaurus sont aussi des ontologies car elles se cantonnent à décrire des liens sémantiques du type "est une sorte de" et son inverse "est représenté par" ou, plus spécifiquement, "est une sous-classe de." Des ontologies plus complexes permettent la

représentation de liens sémantiques plus spécifiques, par exemple, "est localisé dans,". Mais surtout les ontologies les plus abouties permettent également l'intégration de propriétés particulières, de règles d'utilisation et de contraintes.

I.1.2.2. Types de formalismes

On distingue plusieurs types de formalismes pour représenter une ontologie, et différents langages pour chaque type.

a) Les graphes
   i) Les réseaux sémantiques, Topic Maps[4]
   ii) Orienté-Web : RDF et RDF Schéma

b) Logique
   i) Du premier ordre : KIF
   ii) Descriptive : KL-One, OIL, DAML+OIL, OWL

c) Orienté objet : UML + OCL

Afin de comprendre la structuration des ontologies, et la puissance qu'elles peuvent fournir au monde de la représentation des connaissances, nous présentons dans la partie suivante ces langages d'ontologies, et nous consacrons une importante partie pour décrire le langage OWL, ses différents sous-langages, et sa différence avec le langage DAML+OIL.

I.1.2.3 Les langages d'ontologies

a) KIF

KIF [14] est un langage basé sur les prédicats du premier ordre avec des extensions pour représenter des définitions et des méta-connaissances. Tant que la logique du premier ordre est un langage de bas niveau pour l'expression d'ontologies, l'outil *Ontolingua* [15] permet aux utilisateurs de construire des ontologies KIF à un niveau plus élevé de la description par l'importation des définitions des ontologies prédéfinies.

b) KL-ONE

KL-ONE [17] est un langage basé sur la logique de description [16]. Il est une formalisation de représentation de la connaissance à base de cadres[5] (« frame-based »). Ce système maintient la définition des concepts par un simple nommage, et l'indication de la correspondance des concepts dans une hiérarchie de généralisation/spécialisation. De nouveaux termes peuvent être définis par des opérations de conjonction des concepts. Par exemple l'opérateur « *and* » peut être utilisé pour préciser qu'un nouveau concept est une spécialisation commune de plusieurs autres concepts. De nouveaux rôles peuvent être introduits pour représenter les relations qui peuvent exister entre des individus dans le domaine modélisé. Les définitions des concepts peuvent inclure des restrictions sur les valeurs possibles, sur les nombres de valeurs, ou sur le type de valeurs qu'un rôle peut avoir pour un concept.

c) RDF et RDF Schéma

[4] http://www.topicmaps.org/

[5] Un cadre (frame) est un objet nommé représentant un concept générique. Il est doté d'attributs (slots), qui peuvent posséder différentes valeurs (facets), et est destiné à être instancié. FLOGIC est l'exemple le plus connu de langage à base de cadres (frames) [14].

RDF [19] « *Resource Description Framework* » est un formalisme graphique pour représenter des méta-données. Il est basé sur la notion de triplet (sujet, prédicat, objet). Le sujet et l'objet sont des ressources liées par le prédicat. RDF utilise la syntaxe XML, mais il ne donne aucune signification spécifique pour le vocabulaire comme sous classe de, ou le type.

Les primitives de modélisation offertes par RDF sont très basiques. RDF Schéma [19] est un langage qui étend RDF avec un vocabulaire de termes et les relations entre ces termes, par exemple : *Class*, *Property*, *type*, *subClassOf, subPropertyOf*, *range* et *domain*. RDFS est reconnu comme un langage d'ontologie qui définit :

- Des classes et des propriétés.
- Les sous-classes, les super-classes, les sous-propriétés, et les super-propriétés.
- Le domaine de définition et le domaine image des propriétés.

d) DAML + OIL

Beaucoup de travaux ont été faits dans le domaine de la représentation des connaissances parmi lesquels on peut citer les plus importants : SHOE, OntoBroker [22], OIL [23], et encore DAML + OIL [20] qui a remplacé DAML – ONT[6].

DAML + OIL est un langage construit sur des normes précédentes du W3C telles que RDF et RDF Schéma, et étend ces langages avec des primitives de modélisation plus riches. DAML+OIL a été conçu à partir du langage d'ontologie DAML-ONT (DARPA Agent Modelling Language-Ontology, Octobre 2000) en vue de combiner plusieurs composants du langage OIL [23]. OIL[7] « *Ontology Inference Language* » est une représentation basée sur le Web, et une couche d'inférence pour des ontologies. Il combine les primitives de modélisation des langages à base de cadres (frames) avec la sémantique formelle et le raisonnement fournis par la logique de description.

e) OWL

Développé par le groupe de travail sur le Web Sémantique du W3C, OWL peut être utilisé pour représenter explicitement les sens des termes des vocabulaires et les relations entre ces termes. OWL vise également à rendre les ressources sur le Web aisément accessibles aux processus automatisés [18], d'une part en les structurant d'une façon compréhensible et standardisée, et d'autre part en leur ajoutant des méta-informations. Pour cela, OWL a des moyens plus puissants pour exprimer la signification et la sémantique que XML, RDF, et RDF-S [19]. De plus, OWL tient compte de l'aspect diffus des sources de connaissances et permet à l'information d'être recueillie à partir de sources distribuées, notamment en permettant la mise en relation des ontologies et l'importation des informations provenant explicitement d'autres ontologies [8].

(i) Pourquoi OWL ?

XML [11] fournit une syntaxe pour des documents structurés, mais n'impose aucune contrainte sémantique à la signification des documents. RDF est un modèle de données pour représenter les objets (Ressources) et les relations entre eux, fournissant une sémantique simple pour ce modèle qui peut être représenté dans une syntaxe XML. RDF Schema [19] est un langage de définition de vocabulaires pour la description de propriétés et de classes représentées par des ressources RDF. RDFS permet de définir des graphes de triplets RDF,

[6] http://www.daml.org/2000/10/daml-ont.html
[7] http://www.ontoknowledge.org/oil/

avec une sémantique de généralisation / hiérarchisation de ces propriétés et de ces classes. OWL ajoute des vocabulaires pour la description des propriétés et des classes, des relations entre classes (exemple *disjointness*), des cardinalités, des caractéristiques de propriétés (*symmetry*), et des classes énumérées. OWL est développé comme une extension du vocabulaire de RDF et il est dérivé du langage d'ontologies DAML + OIL [20].

(ii) Sous langages de OWL

OWL a trois sous langages de plus en plus expressifs: OWL Lite, OWL DL, et OWL Full :

- **OWL Lite** : supporte les utilisateurs ayant besoin principalement d'une hiérarchie de classification et des contraintes simples (un ensemble est limité à 0 ou 1 élément, par exemple). Il a une complexité formelle inférieure à celle de OWL DL. OWL Lite supporte seulement un sous-ensemble de constructions du langage OWL.
- **OWL DL** : D'après son nom OWL DL utilise la logique de description DL [16]. Il supporte les utilisateurs qui réclament l'expressivité maximale tout en retenant la complétude informatique (toutes les conclusions sont garanties d'être calculables), et la possibilité de décision (les calculs finiront en un temps fini). Il inclut toutes les constructions du langage OWL, qui ne peuvent être utilisées que sous certaines restrictions.
- **OWL Full** : a été défini pour les utilisateurs qui veulent une expressivité maximale et une liberté syntaxique de RDF sans des garanties informatiques. OWL Full permet à une ontologie d'augmenter la signification du vocabulaire prédéfini (RDF ou OWL). Il est peu probable que n'importe quel logiciel de raisonnement soit capable de supporter le raisonnement complet de chaque caractéristique de OWL Full. Autrement dit, en utilisant OWL Full en comparaison avec OWL DL, le support de raisonnement est moins prévisible puisque l'implémentation complète de OWL Full n'existe pas actuellement.

OWL Full et OWL DL maintiennent le même ensemble de constructions de OWL. La différence se situe dans les restrictions sur l'utilisation de certaines de ses caractéristiques et sur l'utilisation des caractéristiques de RDF. OWL permet le mélange libre de OWL avec RDF Schéma et, comme RDFS, n'impose pas une séparation stricte des classes, des propriétés, des individus, et des valeurs de données.

OWL Full peut être vu comme étant une extension de RDF, tandis que OWL Lite et OWL DL peuvent être vus comme des extensions d'une vue restreinte de RDF. Alors les utilisateurs de RDF devraient se rendre compte que OWL Lite n'est pas simplement une extension de RDFS. OWL Lite met des contraintes sur l'utilisation du vocabulaire de RDF (par exemple, *disjointness* des classes, des propriétés, etc.). OWL Full est conçu pour la compatibilité maximale de RDF. Quant à opter OWL DL et OWL Lite, il faut considérer si les avantages du OWL DL/Lite (par exemple, support de raisonnement) l'emportent aux restrictions de DL/Lite à l'utilisation des constructions de OWL et de RDF.

(iii)Différence entre OWL et DAML + OIL

Dans cette partie on se réfère au résumé des changements entre DAML+OIL et OWL présentés dans [21] :

- La sémantique de DAML+ OIL est plus proche de la sémantique du OWL DL.
- Diverses mises à jour sur RDF et RDFS sont incorporées, y compris:
  - Des sous – classes cycliques ne sont pas autorisées

- Des propriétés multiples *rdfs :domain* et *rdfs :range* sont maintenues comme conjonction.
- OWL ne supporte pas l'utilisation des types de données en tant que types:
  <size> <xsd:integer rdf:value="10"/> </size>
  OWL utilise au lieu de cela :
  <size rdf:datatype="http://www.w3.org/2001/XMLSchema#integer"> 10 </size>
- rdf :parsetype = «Collection » remplace rdf :parseType : « daml :collection »
- rdf:List, rdf:first, rdf:rest et rdf:nil remplace daml:List, daml:first, daml:rest, et daml:nil
- daml:item n'est pas maintenu.
- Les restrictions qualifiées sont éliminées du langage, ce qui entraîne à l'élimination des propriétés suivantes : daml:cardinalityQ, daml:hasClassQ, daml:maxCardinalityQ, daml:minCardinalityQ.
- Diverses classes et propriétés sont renommées, comme illustre le tableau suivant :

DAML+OIL	OWL
daml:differentIndividualFrom	owl:differentFrom
daml:equivalentTo	owl:sameAs
daml:sameClassAs	owl:equivalentClass
daml:samePropertyAs	owl:equivalentProperty
daml:hasClass	owl:someValuesFrom
daml:toClass	owl:allValuesFrom
daml:UnambiguousProperty	owl:InverseFunctionalProperty
daml:UniqueProperty	owl:FunctionalProperty

- owl:SymetricProperty, owl:AnnotationProperty, owl:OntologyProperty, et owl:DataRange sont ajoutés.
- owl:DatatypeProperty peut être owl:InverseFuctionalProperty dans OWL Full.
- Des synonymes de classes et propriétés de RDF et RDF Schema sont éliminés, résultant l'élimination de : daml:comment, daml:domain, daml:label, daml:isDefinedBy, daml:Literal, daml:Property, daml:range, daml:seeAlso, daml:subClassOf , daml:subPropertyOf , daml:type, daml:value.
- daml: disjointunionOf est éliminé du langage, puisque elle peut être accomplie en utilisant owl:unionOf ou rdfs:subClassOf et owl:disjointWith.
- Les classes et les propriétés suivantes sont ajoutés pour supporter le versioning: owl:backwardCompatibleWith, owl:DeprecatedClass, owl:DeprecatedProperty, owl:incompatibleWith, owl:priorVersion.
- owl:AllDifferent et owl:distinctMembers sont ajoutés pour spécifier que tous les individus d'une liste font référence à des URIs distinctes.

f) Le modèle orienté objet

Les différentes recherches, au sein de l'intelligence artificielle, pour la représentation de la connaissance ont abouti au développement de divers formalismes pour définir des ontologies, notamment le langage KIF [14] (*Knowledge Interchange Format*) et les langages de représentation de la connaissance dérivés de KL-ONE [17]. Cependant, ces langages de représentation sont peu connus hors des laboratoires de recherche en Intelligence Artificielle.

En revanche, la vaste communauté d'utilisateurs et le support commercial pour les normes orientées objet justifient l'investigation des techniques de modélisation objet pour le développement des ontologies [24]. Par conséquent, on peut exploiter la grande croissance des technologies orientées objet en adoptant UML + OCL comme un formalisme alternatif pour la représentation des ontologies.

(i) UML :

UML est un langage et une notation graphique associée pour l'analyse et la modélisation orientée objet. Le paradigme de modélisation orientée objet est devenu la technique principale dans le développement de logiciel de l'industrie basé sur la vue largement acceptée que la modélisation orientée objet s'adapte bien aux modèles intuitifs du monde [25].

(ii) OCL

OCL (*Object Constraint Language*) est un langage qui permet d'écrire des expressions et des contraintes sur des modèles orientés objet. Une expression est une indication ou une spécification d'une valeur. Une contrainte est une restriction sur une ou plusieurs valeurs sur un modèle orienté – objet ou une partie de ce modèle.

Divers langages de contraintes ont été utilisés dans des modèles orientés objets, par exemple Syntropy [28], Catalysis [29] et BON [30] et des langages de programmation Eiffel [31]. OCL est le langage standard qui fait partie de UML mis par l'OMG comme un standard pour l'analyse et le modèle orienté – objet [32].

Dans les diagrammes des classes, OCL peut être utilisé pour contraindre des valeurs d'attributs et des instances possibles de relations. Les expressions peuvent être utilisées dans un modèle UML afin de :

- spécifier la valeur initiale d'un attribut ou association « *end* ».
- spécifier la règle de dérivation pour un attribut ou une association « *end* ».
- spécifier le corps d'une opération.

Il existe 4 types de contraintes :

- Invariant : est une contrainte qui affirme qu'une condition est satisfaite par toutes les instances de la classe, type, ou interface.
- Précondition : une restriction qui doit être vraie au moment où l'opération va être exécutée.
- Postcondition : une restriction qui doit être vraie au moment où l'opération a fini son exécution.
- Guard : une contrainte qui doit être vraie avant qu'un état de transition soit déclenché.

(iii)Avantages de UML+OCL pour la modélisation des ontologies

L'utilisation d'UML [26], associée à son langage de contraintes OCL [27] présentent les avantages suivantes [24] :

- UML a une communauté d'utilisateurs très large et augmentant rapidement. Les utilisateurs des infrastructures de systèmes d'information distribués seront plus familiers avec cette notation qu'avec les langages KIF et la logique de description.
- Contrairement aux formalismes de la logique de description, UML a une représentation graphique standard qui se trouve importante pour permettre aux utilisateurs des systèmes d'information distribués de naviguer dans une ontologie

et découvrir les concepts qui peuvent exister dans leurs requêtes. En revanche, la logique de description a une syntaxe linéaire mais pas une représentation graphique standard.

- La connaissance représentée en UML est directement accessible pour la compréhension humaine (via sa présentation graphique), et pour le traitement par la machine (via XMI « XML Model Interchange Format » et des librairies de logiciel associés, ou par l'interface de l'application programmeur définie par le MOF[8] « Meta Object Facility » de l'OMG[9] « Object Management Group » [35].
- Un élément de connaissance dans un modèle UML peut être facilement modifié grâce à la nature modulaire de la modélisation orientée objet. Le changement d'une caractéristique dans le modèle n'affecte généralement pas les autres caractéristiques.
- Les modèles UML peuvent être employés dans des buts qui n'étaient pas prévus lors de la création du modèle. Autrement dit, UML est un langage de modélisation abstrait, non attaché à une application particulière.
- Une nouvelle connaissance peut être dérivée des modèles UML par raisonnement à partir de la connaissance existante. En particulier, UML a un langage de contrainte associé OCL qui peut être utilisé pour des éléments dérivés du modèle (qui peuvent être calculés à partir d'autres éléments).

Donc, UML peut être considéré comme un candidat adéquat pour la représentation de la connaissance. En particulier, on se réfère à des diagrammes de classes UML qui fournissent une notation riche pour la définition des classes, de leurs attributs, ainsi que des relations entre elles. Par conséquent, ils peuvent être utilisés pour représenter des ontologies avec une approche orientée objet [35].

Ces différents formalismes de représentation de la connaissance et les différents langages associés ont mené à de nombreuses recherches pour l'intégration des ontologies hétérogènes. Ces travaux font naissance à des approches que nous étudierons dans la partie suivante.

I.2. L'intégration des ontologies

Le mot intégration est utilisé avec plusieurs significations dans le domaine d'ontologie. [41] définie trois différentes significations :

a) *Intégration* : Construction d'une nouvelle ontologie réutilisant (en assemblant, étendant, spécialisant, ou adaptant) d'autres ontologies disponibles. Ces différentes ontologies font partie de la nouvelle ontologie.
b) *Fusion* : Construction d'une ontologie par la fusion de différentes ontologies dans une seule ontologie qui les unifie.
c) *Utilisation* : Construction d'une application utilisant une ou plusieurs ontologies pour spécifier ou implémenter un système à base de connaissance.

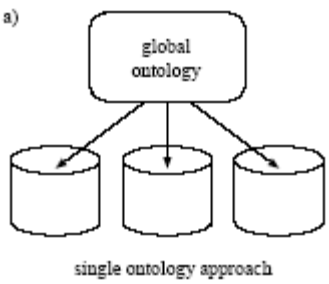
[8] Le MOF « Meta Object Facility » de l'OMG établit un lien entre les méta-modèles dissemblables en fournissant une base commune pour des méta-modèles (voir http://www.omg.org/technology/cwm/)
[9] OMG « Object Management Group » : consortium des compagnies et institutions qui s'occupe de l'ingénierie des logiciels (voir http://www.omg.org/)

La distribution et l'hétérogénéité de l'information ont créé le besoin de l'intégration pour améliorer l'accès aux données et assurer une recherche pertinente. Différentes approches existent et sont évaluées dans [42] vis-à-vis de quatre caractéristiques principales:

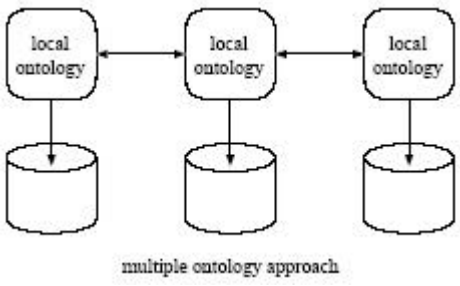
1) Rôle des ontologies

Le rôle et l'architecture des ontologies affectent profondément le formalisme de représentation d'une ontologie. Les ontologies sont utilisées pour la description explicite de la sémantique de l'information source. Ces ontologies sont employées selon différentes méthodes :

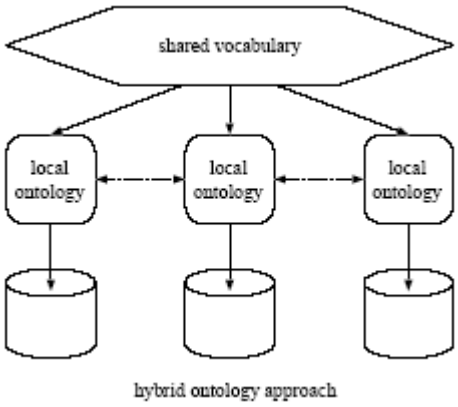
a) ***Approche d'une ontologie simple*** : Cette approche utilise une ontologie globale fournissant un vocabulaire partagé pour la spécification de la sémantique. Toutes les sources d'informations sont reliées à une ontologie globale. Une approche importante pour ce type d'intégration des ontologies est SIMS [43]. Un modèle indépendant de chaque source d'information doit être décrit pour ce système en reliant les objets de chaque source au modèle du domaine global.



b) ***Ontologies multiples*** : Chaque source d'information est décrite par sa propre ontologie. L'avantage est que chaque ontologie source peut être définie sans prendre en considération les autres sources ou les autres ontologies. Mais le manque d'un vocabulaire commun conduit à une difficulté extrême pour comparer différentes ontologies sources. Pour surmonter ce problème, un formalisme de représentation additionnelle définissant le mapping inter-ontologie est fourni. Ce dernier identifie sémantiquement les termes correspondants des différents ontologies sources. Ce mapping est difficile à définir à cause de divers problèmes d'hétérogénéité sémantique qui peuvent se produire. L'approche OBSERVER [44] est utilisée pour ce type.



c) ***Approche hybride*** : Comme l'approche multiple, la sémantique de chaque source est décrite par sa propre ontologie. Mais pour rendre les sources d'ontologies comparables, elles sont construites sur un vocabulaire partagé global [45,46]. Ce vocabulaire partagé comprend les termes de bases ou primitives d'un domaine. Afin de construire des termes complexes, ces primitives sont combinées par des opérateurs, et les termes seront comparés plus facilement qu'une approche multiple. Parfois, ce vocabulaire partagé est représenté sous forme d'ontologie [47]. Dans COIN [45], la description locale d'une information, qu'on appelle contexte, est simplement un vecteur de valeurs d'attributs. Dans METACOTA [46], chaque source d'information est annotée par un label qui indique la sémantique de l'information. Ce label combine les termes primitifs du vocabulaire partagé. Dans BUSTER [47], le vocabulaire partagé est une ontologie générale. Une source d'ontologie est un raffinement de l'ontologie générale. L'avantage de cette approche



est que des nouvelles sources peuvent être ajoutées sans le besoin de la modification du vocabulaire partagé. Elle maintient encore l'acquisition et l'évolution des ontologies.

2) La représentation d'ontologie

On s'intéresse ici aux types de langages utilisés et aux structures générales trouvées. Les systèmes les plus connus utilisent certaines variantes de la logique de description pour représenter des ontologies. Le langage le plus cité est CLASSIC [48] qui est utilisé dans des systèmes tel que OBSERVER [44], SIMS [43]. D'autres langages terminologiques sont GRAIL [49], LOOM [50], et OIL [51] qui est utilisé pour l'intégration terminologique dans l'approche BUSTER [47]. D'autres approches utilisent des extensions qui incluent des règles de base, par exemple le système PICSEL [52] et DWQ [53] utilisant CARIN.

Le principal type de langages utilisés dans des systèmes d'intégration de l'information basée sur les ontologies sont les langages de représentation classiques basés sur les cadres (frame–based) comme ONTOLINGUA [9] et OKBC [54]. Parmi ces systèmes, citons COIN [45], KRAFT [55], Infosleuth [56] et Infomaster [57]. Pour une analyse de la puissance expressive de ces langages, on se réfère à [58] qui évalue les différents langages d'ontologies incluant ceux mentionnés ci dessus.

3) L'utilisation de « mapping »

La relation d'une ontologie avec son environnement joue un rôle essentiel dans l'intégration de l'information. « Mappings » réfère à la mise en correspondance d'éléments d'ontologies entre eux ou avec des éléments d'autres types de ressources sémantiques. Les deux plus importantes sortes de « mapping » utilisées pour l'intégration de l'information sont le « mapping » entre les ontologies et l'information qu'elles décrivent, et le « mapping » entre les différentes ontologies utilisées dans un système.

a) Connexion aux sources d'informations: différentes approches générales sont utilisées pour établir une connexion entre ontologies et sources d'informations:
   i. *Ressemblance de la structure* : une approche directe est de produire une simple copie de structure de la source d'information et l'encoder dans un langage qui rend possible le raisonnement automatique. Cette approche est implémentée par le médiateur SIMS [43] et le système TSIMMIS [59].
   ii. *Définition des termes* : afin de clarifier la sémantique des termes dans un schéma de base de données, des approches comme BUSTER [60] utilisent les ontologies pour définir des termes supplémentaires de la base de données ou de son schéma.
   iii. *Enrichissement de la structure* : combine les deux approches précédentes. Un modèle logique est construit qui ressemble à la structure de l'information source et contient des définitions additionnelles des concepts. Les systèmes qui utilisent l'enrichissement de la structure pour l'intégration de l'information sont OBSERVER [44], Kraft [55], PICSEL [52] et DWQ [53].
   iv. *Méta-annotation* : une nouvelle approche est l'utilisation des méta-annotations pour ajouter la sémantique sur la source d'information. Cette approche est devenue proéminente dans le besoin d'intégrer des informations présentes sur le WORLD WIDE WEB où l'annotation est un moyen naturel pour l'ajout de la sémantique. Des approches développées pour être utilisées sur le WWW sont Ontobroker [61], et SHOE [62].

b) Mapping inter-ontologies: le problème de mapping entre différentes ontologies est bien reconnu dans l'ingénierie de la connaissance. Des approches générales sont utilisées dans les systèmes d'intégration de l'information :
   i. *Les mappings définis* : une approche commune est de fournir la fournir la possibilité de définir des mappings. Elle est utilisée dans Kraft [55] où les translations entre différentes ontologies sont réalisées par des agents médiateurs spéciaux qui peuvent être adaptés pour établir la traduction entre différentes ontologies et même entre différents langages. Cette approche permet une grande flexibilité, mais elle ne peut pas assurer la préservation de la sémantique : l'utilisateur est libre de définir des mappings arbitraires même s'ils n'ont pas de significations ou s'ils produisent des conflits.
   ii. *Relations lexiques* : Ces approches étendent le modèle de la logique de description par des relations inter-ontologies empruntées à la linguistique, par exemple « synonyme », « disjoint », « hyponyme », et « hypernyme » qui sont utilisés dans le système OBSERVER [44].
   iii. *Connaissance de base haut niveau* : afin d'éviter la perte de la sémantique, on doit rester dans le langage de la représentation formelle en définissant des « mappings » entre différentes ontologies. Une méthode directe pour rester dans le formalisme est de relier toutes les ontologies utilisées à une ontologie simple du haut – niveau. Tant que cette approche permet d'établir des connections entre des concepts de différentes ontologies en termes des superclasses communes, elle ne permet pas la correspondance directe.
   iv. *Correspondance sémantique*: pour surmonter l'ambiguïté causée par le « mapping » indirect (comme l'approche précédente), une solution est d'identifier des correspondances sémantiques entre les concepts des différentes ontologies. Ces approches comptent sur un vocabulaire commun pour la définition des concepts des différentes ontologies.

Ces différentes approches traitant de l'intégration des ontologies pour un domaine ont permis la distinction des problèmes d'hétérogénéité entre les différentes ontologies. Nous avons consacré la partie suivante pour étudier les types de discordance entre les ontologies tant au niveau de leur structure qu'au niveau de leur sémantique.

I.3. Hétérogénéité

La modélisation des connaissances, et le partage de données donnent naissance à des ontologies. Cependant, beaucoup de projets opérationnels et de recherche ont abouti à la création de multiples ontologies, parfois pour un même domaine. Ces ontologies ne sont homogènes ni dans leur structure, ni dans leur sémantique, et l'hétérogénéité est aperçue lors de la mise en commun de la connaissance représentée avec ces ontologies.

Un certain nombre de recherches tente à étudier les causes du problème d'inconsistance ou de discordance entre différentes ontologies. Ceci a amené à la mise en œuvre de multiples approches qui visent à traiter ce type de problème. Ces approches mettent en considération les différents cas qui peuvent y exister.

Wiederhold [36] a distingué dans son approche les cas où la consistance, pour un domaine, n'est pas maintenue. Il a référé les différences entre des ontologies aux différences de la nomination et de la portée en terme des noms et de la sémantique des méta-informations sur les attributs qui apparaissent dans le schéma, et des noms et la sémantique qui

apparaissent comme des valeurs dans le contenu de la source d'information. Quatre différences y sont déterminées :

1- Deux items d'attributs dans deux ontologies ont deux noms différents est le cas le plus commun et le plus facile à résoudre. Une simple table peut être utilisée pour maintenir la correspondance désirée, et rassembler l'information.
2- La différence de portée des concepts manipulés est plus insidieuse, et doit être déterminée par une analyse du contenu. La résolution exige l'établissement, la validation, et le traitement des règles.
3- Des différences relatives aux unités de mesure prises par des nombres sont communes. Une conversion peut être établie par une formule (un mètre = pieds/0.305). Dans ce cas des règles ou des tables doivent être introduites.
4- Des portées sur des attributs sont subjectives. Le terme «*chaud*» a une signification différente dans le domaine du temps de celle dans le domaine des moteurs des véhicules. Les différences dans la portée mènent aux différences dans la mise en référence qui fait leur résolution pourtant plus critique.

L'approche Mitra [37] traite l'hétérogénéité sémantique entre multiples sources. Afin d'établir une correspondance entre les termes des ontologies, cette approche explore les discordances et leurs solutions en utilisant des graphes objets. Quatre types de discordance sont déterminés :

1- Discordance de la sémantique des termes : si un terme de deux ontologies différentes réfère à des concepts différents.
2- Discordance de structure : un même terme dans une source correspond à plusieurs termes dans une autre source, et cause qu'un nœud dans un graphe s'approprie à plusieurs nœuds dans un autre. Cette approche permet ce type de correspondance.
3- Discordance des instances : une instance d'une classe dans une source n'est pas une instance de la même classe d'une autre source. Les règles d'articulation doivent spécifier explicitement quelles correspondances on veut générer dans ces cas.
4- Discordance de granularité : si on a deux nœuds correspondants, et le grand-père d'un nœud correspond avec le père de l'autre. Autrement dit, dans le premier graphe un concept est organisé dans une hiérarchie plus élaborée que dans le deuxième graphe.

Le besoin de traduire une représentation symbolique d'une connaissance en différents formats était le motif de la création du système « *Ontomorph* » par Chalupsky [38]. Ce système permet de traiter les problèmes qui peuvent se produire lors d'une telle traduction. Le scénario est de traduire les différents modèles du même domaine. Ces modèles se diffèrent à travers de multiples dimensions. Selon Chalupsky, six types de discordances communes qui peuvent être trouvées sont identifiés :

1- Syntaxe du langage de représentation de la connaissance (KR) : deux ontologies peuvent avoir des syntaxes et des conventions syntaxiques différentes.
2- Expressivité du langage de KR : chaque langage supporte l'expressivité d'une manière différente des autres langages. Par exemple, la négation qui peut être représentée par un langage mais pas dans un autre. Dans ce cas, des choix difficiles doivent être pris sur la manière de traduire les idiomes représentatifs.
3- Conventions de modélisation : même si le langage de représentation de la connaissance pour une base de connaissance source et, une autre cible sont les mêmes, ça n'empêche pas d'avoir des différences à cause de la manière de modélisation d'un domaine particulier. Par exemple, la classe personne peut avoir deux sous-classes « male » et « femelle », ou on peut ajouter une relation « sexe » pour déterminer le sexe de la personne.

4- Modèle de recouvrement et granularité : Des modèles diffèrent par leur recouvrement d'un domaine particulier, et la granularité avec laquelle des distinctions sont faites. C'est la raison la plus forte pour la fusion des ontologies. Par exemple, une ontologie modélise les voitures mais pas les camions, une autre représente les camions sous des catégories.
5- Paradigmes de représentation : différents paradigmes sont utilisés pour représenter des concepts comme le temps, l'action, le plan, la causalité.
6- Systèmes d'inférence : une raison pour que des modèles semblent différents est due à leur construction pour produire des inférences désirées avec un moteur d'inférence particulier.

Enfin, nous constatons la classification des types d'hétérogénéité entre les ontologies en deux classes : les discordances liées à la syntaxe, et celles liées à la sémantique. Ces dernières sont beaucoup plus complexes, et leur résolution nécessite de grandes recherches. Cependant, cette approche ne prend pas en considération l'étude du langage OWL.

I.4. Synthèse

L'analyse bibliographique réalisée dans ce chapitre présente le domaine de la représentation de la connaissance. La sémantique définie dans l'introduction de ce chapitre nous a permis de constater son importance dans le partage des données, surtout avec l'abondance de ces données représentées sur le Web. L'ontologie est présentée comme un outil essentiel pour représenter le monde réel par la conception des objets de ce monde dans un modèle représentatif, décrit par les classes y existantes, et les propriétés qui définissent les relations établies entre ces objets. La définition des contraintes dans une ontologie a explicité la sémantique des données, et par conséquent aboutit à son traitement par la machine.

Les différents formalismes possèdent de niveaux d'expressivité différents. Le standard OWL s'est montré capable de jouer un rôle du pivot, grâce à sa compatibilité avec d'autres langages DAML+OIL, RDF, et RDFS. Le modèle orienté objet s'est révélé capable de représenter une ontologie grâce au langage de modélisation UML associé avec son lagage de contrainte OCL. Les approches d'intégration des ontologies nous ont apporté des points forts sur l'étude structurelle et les mécanismes de « mapping ». Cette intégration cause l'apparition des problèmes d'hétérogénéité entre différentes ontologies dans un même domaine. L'étude de ces types de problèmes nous a rendu conscient des difficultés que nous pouvons rencontrer durant notre travail.

Pour conclure, nous pouvons constater que l'unicité de format de représentation de la connaissance est un mécanisme qui facilite l'intégration des différentes ontologies, et sert à améliorer la communication entre les différents acteurs dans un même domaine. Dans le chapitre suivant, nous proposons une approche pour l'intégration des ontologies dans le domaine de la construction des bâtiments, en appuyant notre approche sur une application concrète.

CHAPITRE II

PROPOSITION D'UNE METHODOLOGIE D'ALIGNEMENT D'ONTOLOGIES HETEROGENES

La diversité de formats de représentation de la connaissance pose le problème de l'interopérabilité dont nous avons parlé dans la première partie de notre travail. Ce problème affecte le processus de communication, et par conséquent retarde le partage des données sur le Web. Dans notre état de l'art, nous avons constaté un nombre important de recherches qui visent à traiter ce type de problèmes. Nous avons consacré une partie importante aux ontologies, aux différents langages, et en particulier au langage d'ontologies OWL, recommandation du W3C, depuis février 2004.

En partant de l'importance de OWL, et afin de traiter les différents problèmes posés par la différence de formats, nous proposons une démarche qui s'appuie sur les deux étapes suivantes :

1) La première étape est fondée sur le mécanisme de conversion de différents formats en un format pivot. Il s'agit de traduire les différents formalismes de représentation de connaissances vers le langage OWL.

2) Dans la deuxième étape, nous proposons un mécanisme de définition de règles de « *mapping* » qui permettent les correspondances entre les ontologies en OWL créées lors de la première étape, tout en respectant l'expressivité des langages d'origine.

Ce travail s'appuie plus sur la traduction des formats des langages que sur l'étude des contenus des ontologies. Cette dernière étape, qui s'appuie sur une étude de pertinence contextualisée des règles de *« mapping »* est le sujet d'un autre stage de DEA réalisé dans l'équipe en parallèle et en collaboration avec celui-ci.

II.1. Conversion vers un format pivot

Le mécanisme de conversion présenté ici vise à homogénéiser différents formats de représentation des connaissances décrits précédemment. Il s'appuie sur une distinction explicite entre les niveaux de représentation de la connaissance. Il s'agit de faire une analyse de la structure de leurs données et de leurs modèles afin d'identifier les différences de modélisation et de conceptualisation. Nous identifions trois différents niveaux d'abstraction :

1) Les méta-modèles : c'est la couche du plus haut niveau. Elle représente la forme structurelle d'un langage de représentation de connaissance. Cette couche est indispensable pour connaître la puissance de langage à modéliser, la manière de représenter les concepts et les relations entre eux. C'est à ce niveau que la grammaire du langage est définie. En ce qui concerne les langages, nous avons considéré les langages suivants : XML pour les taxinomies, RDF pour les langages permettant de définir des graphes, et RDF-S et DAML+OIL en tant que langages de définition d'ontologies. À ce niveau, l'étude faite est surtout théorique car elle contient une vue générique sur ces langages de représentations et sur le mécanisme de transformation de leurs méta-modèles en celui d'OWL.
2) Les modèles : c'est le niveau de description des données représentées. Chaque modèle est spécifique à un cas particulier de représentation de données. On peut trouver les classes qui peuvent y exister, les attributs de chaque classe, et les relations existantes entre les différentes classes. Tous ces modèles sont représentés dans le langage de modélisation UML, qui nous a servi de référence pour la modélisation durant tout notre travail. Pour illustrer notre propos, nous avons travaillé sur une application dans le domaine de la construction. Pour cela, nous avons utilisé ou élaboré un modèle spécifique pour chacun des trois projets suivants : nous avons utilisé un modèle déjà établi pour le projet *bcXML*, nous avons conçu un modèle pour le projet *Edibatec*, et nous avons représenté les super concepts du projet *eCognos* dans un modèle spécifique.
3) Les données : c'est le niveau le plus bas qui regroupe les instances ou les données représentées. La partie théorique n'est pas concernée par ce niveau d'abstraction. En pratique, nous avons implémenté notre approche par des algorithmes qui servent à réaliser la conversion des documents instances de nos projets vers des ontologies OWL.

De façon à rester générique, nous avons réalisé la partie théorique de notre travail uniquement au niveau de la conversion des méta-modèles, puisque c'est le niveau des langages. Par contre, au niveau des modèles et des données, le travail est spécifique à l'application.

Cette partie est consacrée à une vue générique des méta-modèles de différents formalismes faisant partie de notre travail, et cités dans notre approche au-dessus. Nous abordons par la conception du schéma méta-modèle OWL, et la définition de différents éléments présents dans ce schéma.Ensuite, nous continuons par l'expression des méta-modèles des langages de représentation, et le mécanisme de conversion de chacun de ces méta-modèles vers OWL. Ces langages sont XML pour décrite une taxonomie, le langage RDF pour les représentations graphiques, et les langages RDF-S et DAML + OIL pour les langages d'ontologies.

II.1.1. Le Méta-modèle pivot (OWL)

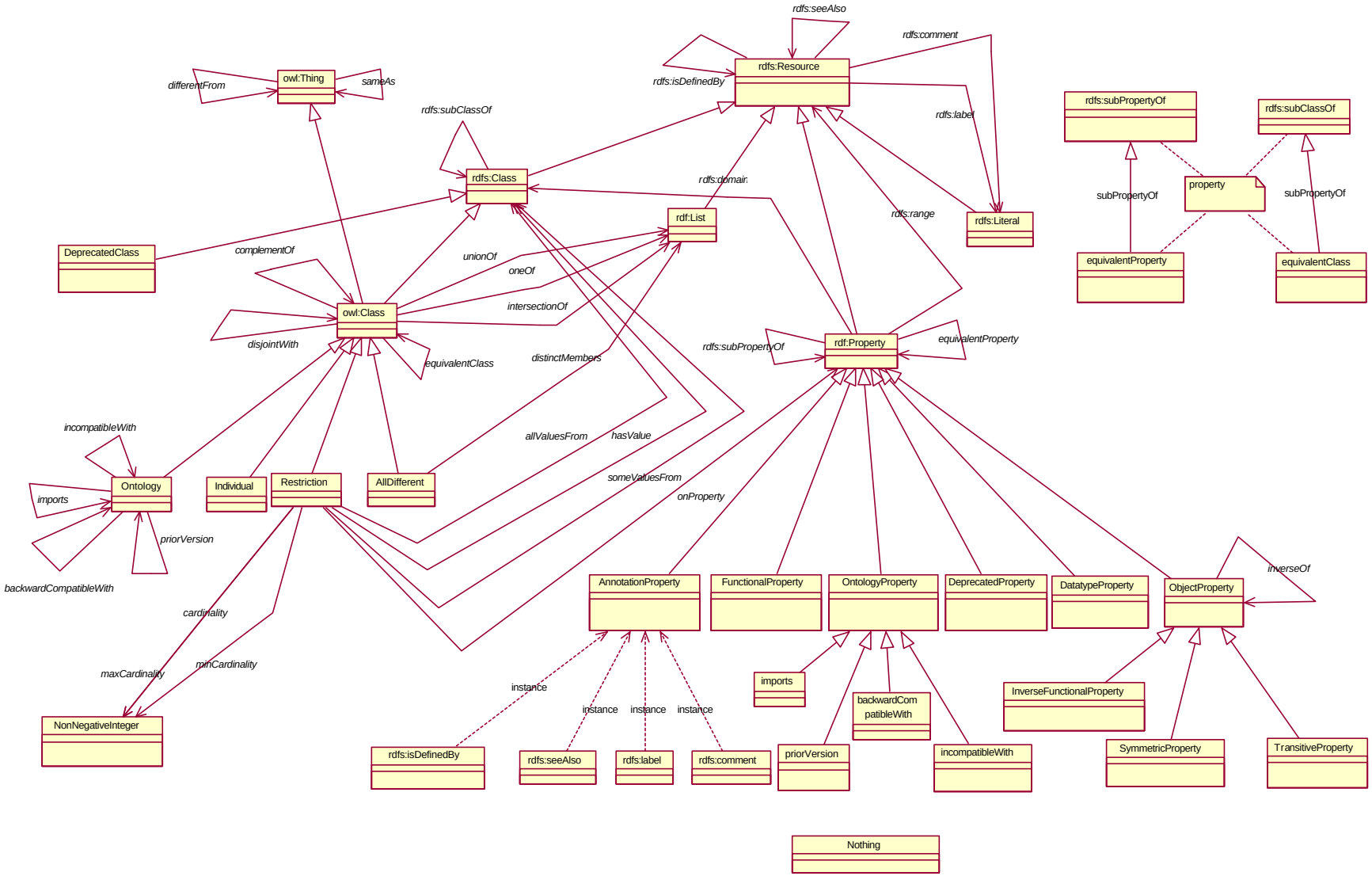
Comme présenté dans notre état de l'art, le langage OWL fournit les trois sous-langages suivants : OWL Lite, OWL DL, et OWL Full. OWL Lite a été établi pour une implémentation facile, et permet la définition des contraintes simples. Par exemple, les valeurs d'une cardinalité sont restreintes à 0 et 1. OWL DL, est conçu pour supporter la logique de description. OWL Full contient les mêmes constructions que OWL DL. La différence apparaît dans l'utilisation de ces constructions : OWL DL exige une séparation des classes, des propriétés, des individus, et des valeurs de données. OWL DL permet le développement de systèmes de raisonnement qui ont besoin des contraintes exigées par OWL DL. Les caractéristiques de OWL DL nous conduisent à adopter ce langage comme pivot pour notre travail.

OWL est le langage standard qui joule le rôle de cible pour notre approche, et dans lequel les différentes ressources doivent être représentées. La conception d'un diagramme de classes UML du méta-modèle d'OWL est une étape essentielle pour identifier les concepts de ce langage, et par conséquent sa puissance pour représenter les données. Ce diagramme est présenté en figure II.1. Il permet d'avoir une vue globale des différentes classes et les différentes relations qui existent entre ces classes en OWL.

Dans ce schéma, nous représentons les classes qui peuvent décrire une ontologie, et les relations entre elles. La classe « *owl:Thing* » est la classe de tous les individus, et la super classe de toutes les classes de OWL. « *owl:Nothing* » est une classe qui ne peut pas être instanciée, et elle est la sous-classe de toutes les classes. Dans une ontologie OWL, il y a la possibilité de hiérarchiser différentes propriétés. Cette hiérarchisation ne peut pas être représentée avec le diagramme de classes UML. Pour cela, nous représentons ces propriétés par des classes, et nous indiquons la relation « owl:subPropertyOf » entre ces propriétés. Par exemple, dans le schéma, la propriété « owl:equivalentProperty » est une sous-propriété de la propriété « owl:subPropertyOf ».

Dans la suite, ce schéma sert de représentation cible pour les mécanismes de transformation des autres ressources, dont les [méta]-modèles sont également exprimés en UML. Les paragraphes suivants présentent donc une étude de l'expression des autres formalismes par rapport à notre méta-modèle OWL. Pour cela, nous partons des formalismes les plus simples, les structures hiérarchiques (ou taxonomies), puis nous nous intéressons aux formalismes à base d'arcs, ou de relations (les langages de définition de graphes), pour enfin traiter les hybridations de ces deux types que sont les langages de définition d'ontologies.

Figure II. 1 : Le méta-modèle OWL.



II.1.2. Les taxonomies (applications XML)

D'après la définition d'une taxonomie (cf. partie précédente), une taxonomie permet d'organiser un document sous une forme hiérarchique. Une taxonomie est un ensemble d'éléments (i.e. de concepts), reliés entre eux par un seul type de relation (en général, des relations de composition).

Il est important de noter à ce stade que la présence de plusieurs types de relations (héritage et agrégation dans un diagramme de classes, liens « voir aussi » dans un thésaurus) dans une même ressource, fait « sortir » cette ressource de la définition d'une taxonomie. Cette ressource devient alors un graphe, et le mécanisme de conversion du typage des relations est présenté au paragraphe suivant.

Pour définir la structure générique d'une taxonomie, nous considérons le méta-langage de représentation de données XML, qui est un outil suffisamment puissant pour représenter, dans sa syntaxe de base, n'importe quelle taxonomie. Un document XML est formé d'une imbrication d'éléments. Un élément possède nécessairement un nom (éventuellement qualifié), éventuellement un ou plusieurs attributs et éventuellement un contenu. Un attribut possède toujours un nom (éventuellement qualifié) et une valeur. Pour simplifier, une valeur d'attribut peut être soit du texte libre, soit un type de données particulier (ID, référence à un ID, entité…). Le contenu d'un élément peut être constitué d'un ou plusieurs éléments et d'une ou plusieurs parties textuelles.

Cette structure étant relativement simple, le mécanisme de conversion en OWL l'est également. Nous proposons les éléments suivants pour la conversion des taxonomies en OWL :

- ✓ Les concepts existants dans une taxonomie peuvent être représentés dans une ontologie par des classes (syntaxe OWL : « *owl:Class* »).
- ✓ Les attributs de ces concepts peuvent être traités comme des propriétés dans une ontologie (syntaxe : « *owl:Property* »). Dans ce cas, le texte libre devient un littéral RDF, et les valeurs contrôlées des ressources définies sur un domaine particulier.
- ✓ La relation de composition entre les concepts peut être représentée avec OWL grâce à la propriété « *rdfs:subClassOf* ».

Pour représenter une taxonomie dans une ontologie OWL, nous pouvons restreindre le méta-modèle OWL à un niveau où nous restons capable de représenter les concepts et les attributs de cette taxonomie. Ce modèle retreint apparaît dans la figure 2. Notons que ce modèle est également valide en OWL Lite.

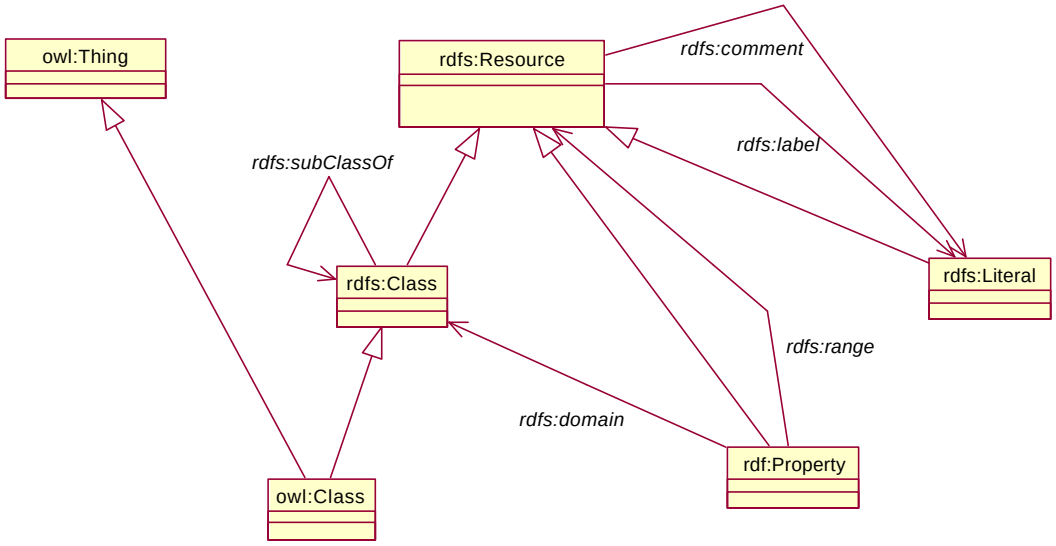


Figure II. 2 : Modèle OWL restreint à une taxonomie

II.1.3. Les langages de définition de graphes (RDF)

Les graphes qui nous intéressent ici sont des structures de connaissances composées d'arcs, un arc étant une relation entre des concepts. Nous ne considérons ici que les arcs orientés binaires. Les arcs bidirectionnels sont considérés comme deux arcs distincts. On remarque qu'une taxonomie peut être représentée par un graphe.

Le langage RDF permet de décrire des arcs correspondant à cette définition. RDF est basé sur la notion de triplet (*Sujet, Prédicat, Objet*). Le *sujet* a une propriété nommée *prédicat* avec la valeur *objet*. Ces trois composantes d'un arc peuvent être définies comme des ressources. Une ressource peut faire partie de plusieurs triplets et avoir des rôles différents. Un ensemble de triplets reliés entre eux par des ressources constitue un graphe RDF.

Le seul méta-modèle existant pour RDF étant le triplet, nous n'avons pas conçu de diagramme UML pour décrire ce type de structure. Pour faire la conversion en OWL, il faut représenter les entités suivantes :

- Un sujet peut être représenté dans une ontologie OWL par une classe (syntaxe OWL : « *owl:Class* »).
- Un objet peut être représenté soit par une classe (syntaxe OWL : « *owl:Class* »), soit par un élément textuel ou *littéral* (syntaxe : « *rdfs:literal* »).
- Un prédicat est une propriété qui est représentée soit par « *owl:ObjectProperty* » si l'objet est une ressource, ou par « *owl:DatatypeProperty* » si l'objet est de type littéral. Les deux instructions « *rdfs:domain* » et « *rdfs:range* » permettent de spécifier respectivement le sujet et l'objet du triplet.

II.1.4. Les ontologies (RDFS et DAML+OIL)

Par rapport aux langages de description de graphes, les langages d'ontologies fournissent un vocabulaire permettant d'étendre la sémantique des graphes par l'ajout de constructions de haut niveau. Par exemple, il est possible de spécifier ainsi que deux classes sont disjointes ou qu'une propriété est transitive. De plus, une ontologie peut contenir des contraintes reliant les concepts.

Différents langages d'ontologies ont été étudiés dans la partie précédente. Nous ne nous intéressons ici qu'aux langages orientés Web (RDF-S, DAML+OIL et OWL). Faute de temps, nous n'avons pas considéré dans ce travail les langages orientés objet (UML+OCL), les langages basés sur d'autres paradigmes, comme la logique du premier ordre (KIF) ou les Topic Maps.

RDFS étend RDF par un vocabulaire permettant la structuration des méta-données (par exemple, « *rdfs:subClassOf* ») et l'expression de contraintes sur les ressources (via la propriété « *rdfs:ConstraintProperty* »). Le méta-modèle de OWL hérite des caractéristiques des classes propres au langage RDFS. Par exemple « *owl:Class* » dérive de « *rdfs:Class* ». Ceci rend la relation entre RDFS et OWL semblable à celle définie entre RDF et RDFS. Autrement dit, OWL peut être considéré comme une extension de RDFS par la définition d'une sémantique pour des ressources spécifiques du vocabulaire RDFS. OWL peut créer une sémantique qui n'est pas fournie par RDFS, par exemple « *owl:FunctionalProperty* ». Un document valide selon un schéma RDFS peut alors être considéré comme une ontologie OWL Full. Convertir un document RDFS en OWL DL revient donc à transformer une ontologie OWL Full en OWL DL. Ce problème est traité plus loin dans cette partie.

DAML+OIL est le langage d'ontologies prédécesseur de OWL. Quand ce dernier a été conçu, il avait pour base le langage de DAML+OIL. Cela nous permet de déduire une certaine similarité entre les deux langages malgré un nombre significatif de changements. Dans notre état de l'art, nous avons consacré une importante partie pour décrire le langage DAML+OIL, et distinguer la différence syntaxique qui existe en comparaison avec OWL. Il apparaît que le travail de conversion de DAML+OIL vers OWL a déjà été fait dans la littérature. *A priori*, cette conversion ne comporte pas de contrainte spécifiant le respect du sous-langage OWL DL. Par conséquent, cette transformation devrait produire des ontologies en OWL Full. Cependant, DAML+OIL étant basé sur la logique de description, si la conversion est « bien » faite, les ontologies produites doivent nécessairement respecter la sémantique OWL DL. En outre, cette étude nous permet de déduire que le méta-modèle du langage DAML+OIL peut être facilement conçu à partir de celui de OWL tout en modifiant les classes renommées, ajoutant les classes additionnelles, et supprimant les classes qui ont été rejetées. Ce méta-modèle n'étant pas nécessaire pour notre travail, il n'a pas été réalisé.

II.2. Mise en correspondance des ontologies

Nous avons vu dans l'état de l'art réalisé précédemment, qu'il existe plusieurs moyens d'associer des connaissances appartenant à plusieurs ontologies. Nous en avons choisi une qui consiste à permettre la mise en relation de concepts et de propriétés entre différentes ontologies générées en OWL DL, et d'indiquer la relation qui existe entre eux.

Seuls les experts du domaine représenté sont capables de spécifier de telles relations. Nous nous limitons donc à fournir un mécanisme de définition et une structure permettant le

stockage de telles relations. Une telle structure consiste en une nouvelle ontologie représentant ces relations en OWL DL, que nous appelons « ontologie de correspondance » (en anglais : *« articulation ontology »*). Les paragraphes suivants détaillent certaines relations entre les concepts qui seront intégrées dans cette ontologie.

Pour intégrer les différentes ontologies, il est nécessaire de pouvoir indiquer qu'une classe ou une propriété particulière dans une ontologie est équivalente à une autre classe ou propriété d'une autre ontologie. Dans la syntaxe de OWL, on a la possibilité de définir des relations d'équivalence entre les classes grâce à la construction « *owl:equivalentClass* » pour indiquer que deux classes ont exactement les même instances, ou **individus**. De même, la construction « *owl:equivalentProperty* » permet de faire correspondre deux propriétés reliant un individu à un même ensemble des autres individus.

Notons que les différents sous-langages de OWL traitent différemment la distinction entre classe et individus de classes. Dans OWL DL, les classes dénotent des ensembles des individus ; mais elles ne peuvent pas être les individus elles-mêmes. Par contre, en OWL Full, une classe peut être traitée comme étant une instance. Par conséquent, en OWL Full « *owl:sameAs* » peut être utilisé pour indiquer que deux classes sont identiques.

Le mécanisme d'identité entre les individus (syntaxe : « *owl:sameAs* ») est proche de celui d'équivalence de classes, mais il stipule que deux individus sont identiques. C'est le cas quand des individus ayant des noms différents réfèrent au même individu. Par exemple, deux concepts dans des ontologies différentes peuvent désigner une même ressource. Dans ce cas, ils sont identiques.

Notons que la mise en correspondance par « *owl:sameAs* » utilisé pour spécifier l'égalité de deux classes n'est pas la même que celle obtenue avec « *owl:equivalentClass* », qui n'implique pas l'égalité de classe, c'est-à-dire ne dénote pas le même concept. L'égalité réelle exprimée entre deux classes grâce à « *owl:sameAs* » n'est possible qu'avec OWL Full, qui traite les classes comme des individus.

Pour indiquer une différence entre les individus (relation opposée de « *owl:sameAs* »), la construction « *owl:differentFrom* » permet d'exprimer que deux individus sont distincts. Un mécanisme commode existe pour définir que tous les individus d'un ensemble sont mutuellement distincts : « *owl:AllDifferent* ».

Les relations décrites ici restent le fait des experts, supposés décider de leur établissement. Les conditions exprimées ci-dessus pour décrire ces relations ne sont que nécessaires – et donc non suffisantes – à leur définition. D'autres relations, dont la sémantique peut être définie par les experts peuvent également être intégrées dans l'ontologie de correspondance. Toutefois, l'application décrite dans la section suivante ne prend pas en charge toutes ces relations. Elle se limite à l'établissement de relations d'équivalence de classes.

II.3. Application

Au niveau des modèles et des données, la conversion vers une ontologie OWL est spécifique. Nous prenons à ces deux niveaux les trois ressources : bcXML, e-Gognos, et Edibatec. Après l'étude de ces ressources, une analyse de structure permet d'identifier les différences de modélisation et de conceptualisation. La conversion de ces ressources vers une ontologie OWL DL est réalisée par les deux étapes suivantes : (voir figure II.3)

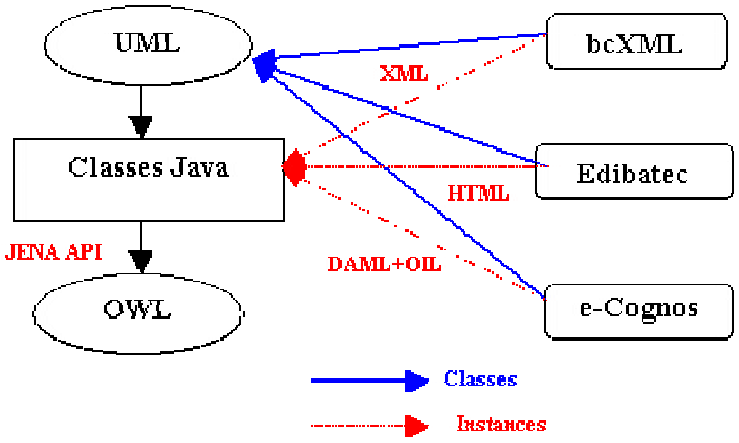


Figure II. 3 décrivant les étapes de conversion

1) Elaboration des modèles : durant cette étape, nous nous intéressons à la conception des modèles spécifiques et adéquats pour chaque projet, et nous proposons un mécanisme de transformation des concepts en des classes JAVA. Les modèles sont exprimés en UML, sous formes des diagrammes de classes.

2) Implémentation du prototype : dans la couche de données les instances des trois ressources citées ci-dessus sont représentées dans différents langages. Nous visons à réaliser par la programmation en JAVA la conversion de ces différents formats en langage d'ontologie OWL. Dans ce but, nous nous servons de l'API Jena que nous détaillons dans la partie implémentation.

II.3.1. Modèles

La première étape pour appliquer notre approche est d'élaborer un modèle pour chaque ressource, et expliquer le processus de conversion des classes présentées dans ces modèles en des classes JAVA. Pour chaque ressource, nos décrivons ses caractéristiques, et ensuite nous présentons l'étude de son modèle. Notons que les trois ressources traitées bcXML, e-Cognos, et Edibatec sont présentées dans l'ordre chronologique.

II.3.1.1. bcXML

bcXML définit un cadre pour la présentation d'un vocabulaire basé sur XML pour l'industrie du bâtiment et de construction. Ce vocabulaire est un ensemble des notions qui donne une signification à des messages XML reliés au domaine. Il fournit pour l'industrie européenne de BC une infrastructure de communication puissante et de prix réduit qui :

- supporte la communication électronique « eBusiness » entre les clients, les architectes et ingénieurs, et les fournisseurs des produits, des composants et des services.
- supporte le multi langage.
- peut être intégrée avec des applications eCommerce.

La modélisation qui permet à une taxonomie d'être définie est réalisée par un méta-schéma XTD (*eXtensible Taxonomy Definition*) présenté dans la figure II.4. Ce modèle représente la couche la plus abstraite pour représenter la taxonomie de bcXML.

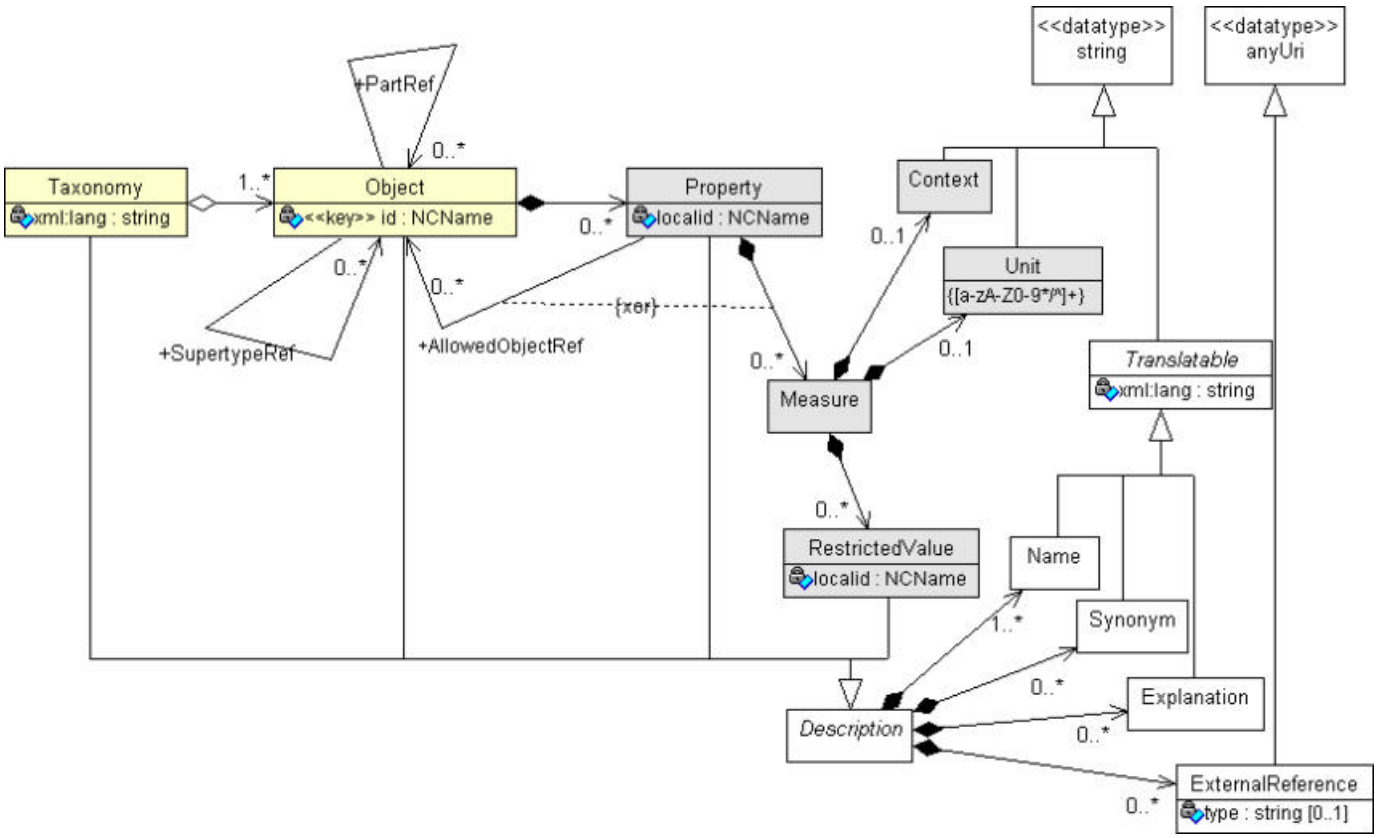


Figure II. 4 : Modèle bcXML

La classe «*Description* » est une définition textuelle d'un terme pour l'interprétation humaine et qui permet le processus de translation. Cette classe identifie, en différents langages, la liste des noms, des synonymes, des explications, et des références externes. Elle est utilisée comme super-classe pour décrire une taxonomie, un objet, une propriété, et une

valeur restreinte d'une mesure. Une taxonomie a un attribut langage, et est formé par une séquence d'objets. Chaque objet a un attribut clé pour l'identifier. Il est en relation avec ses objets composants, et les objets qui le composent. Un objet est lié à une série de propriétés.

Chaque propriété a un identificateur local, et elle est formée par un choix entre une série de mesures, ou d'objets de référence permis. Chaque mesure est définie par une unité de mesure, un contexte, et une série de valeurs restreintes. Chaque valeur restreinte est identifiée par un identificateur local.

Dans ce modèle la classe « *Translatable* » permet d'ajouter un attribut de langage à une chaîne de caractères. La classe « *ExternalReference* » permet de faire un lien avec un document externe. On peut indiquer le type de document par l'attribut optionnel « *type* ».

II.3.1.2. e-Cognos

e-Cognos a une infrastructure basée sur le Web inclut des services permettant de créer, capturer, indexer, rechercher et disséminer la connaissance. La méthode adoptée pour définir le niveau le plus élevé de concepts dans l'hiérarchie de l'ontologie était la suivante :

*« In the context of a **Project**, a group of **Actors** uses a set of **Resources** to produce a set of **Products** following certain **Processes** within a work environment (**Related Domains**) and according to certain conditions (**Technical topics**) ».*

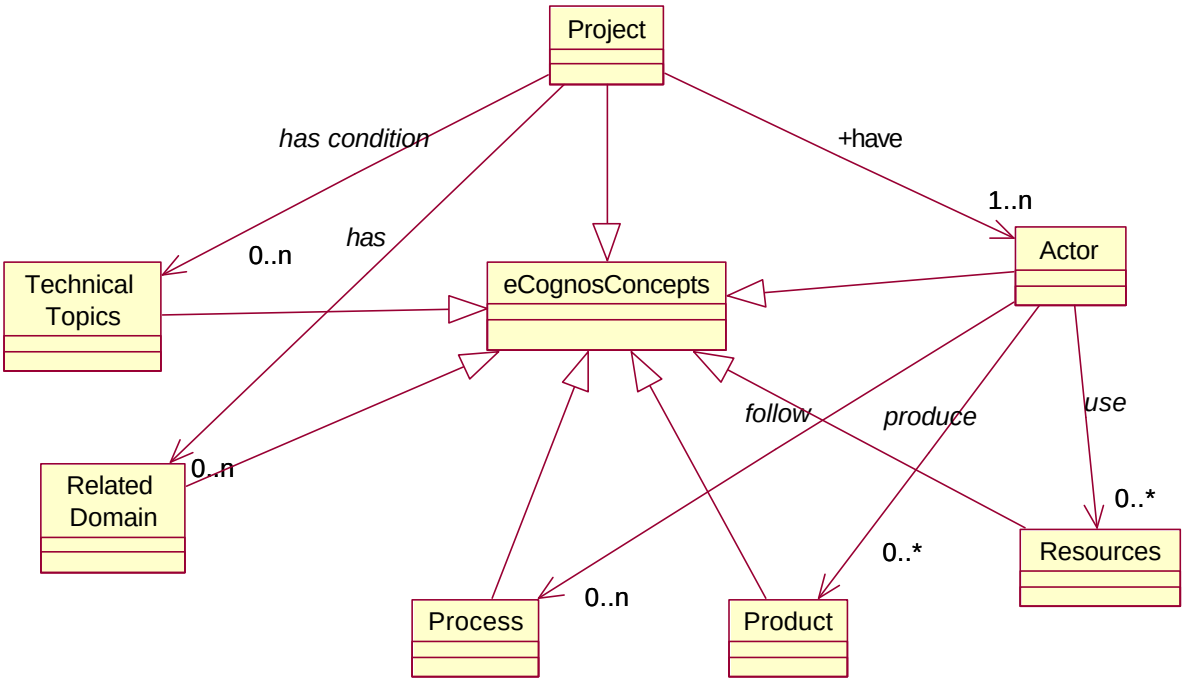


Figure II. 5 : Les super concepts de la ressource e-Cognos

Cette méthode est représentée dans la figure II.5. Dans cette figure, les concepts de plus haut niveau de l'ontologie e-Cognos sont représentés dans la classe « *eCognosConcepts* ». De cette classe dérive toutes les autres classes de la taxonomie. Le

projet est représenté par la classe « Project ». Cette classe est reliée à un groupe d'acteurs « *Actor* » qui travaillent dans le contexte de ce projet. Ces acteurs utilisent plusieurs ressources « *Resources* » pour produire un ensemble de produits « *Product* ». Plusieurs processus «*Process* » se présentent pour la réalisation de la production. Le projet se situe dans un environnement de travail « *Related Domain* », et soumis à certaines conditions « *Technical Topics* ».

II.3.1.3. Edibatec

Dans le cadre des échanges entre installateurs et fournisseurs, l'objet d'Edibatec est de proposer une taxonomie commune permettant aux différents fournisseurs de mettre en commun leurs catalogues de produits et de composants techniques. L'installateur dispose ainsi sous une forme compacte (CD-ROM) de toutes les informations techniques et commerciales qui lui sont nécessaires dans les différentes phases de conception, de réalisation et de maintenance d'une installation de génie climatique ou électrique. Cette taxonomie est également accessible gratuitement sur le Web, en différents formats (HTML, Excel).

La taxonomie Edibatec rassemble actuellement l'ensemble des caractéristiques techniques et commerciales des produits dans des domaine du génie électrique et climatique/sanitaire. Il est prévu de l'étendre à d'autres domaines. Nous avons réalisé un modèle détaillé des classes et de leurs propriétés de cette taxonomie, présenté en figure II.6.

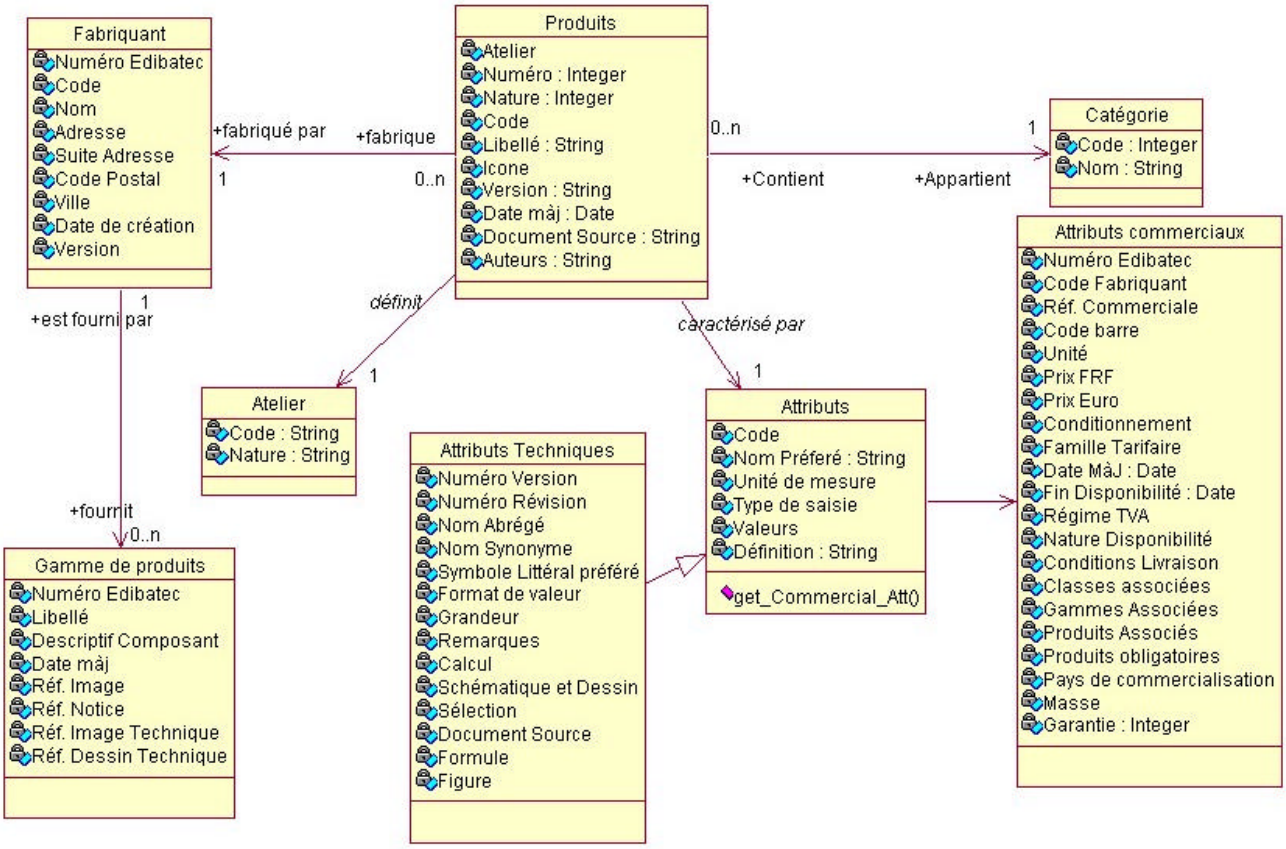


Figure II. 6 : Le modèle de la ressource Edibatec.

Ce schéma définit les différents concepts qui existent dans ce catalogue, les attributs qui les caractérisent, et les relations entre elles. Les attributs commerciaux caractérisent d'un

point de vue commercial chaque produit. Ces attributs sont décrits avec un code, un nom préféré, une unité de mesure, un type de saisie, des valeurs (en cas de type énumération), et une définition. Puisque les attributs commerciaux sont communs à tous les produits, nous avons défini ces attributs dans une classe « Attributs Commerciaux » qui regroupe tous ces attributs.

Concernant les attributs techniques, ils sont propres à chaque produit, et ils sont décrits par des caractéristiques spécifiques mentionnées dans la classe « Attributs Techniques », en plus que celles héritées de la classe « Attributs ». Il existe une relation d'héritage entre les deux classes précédentes.

II.3.2. Implémentation du prototype

Dans cette partie, nous utilisons les données présentées dans les trois ressources bcXML, e-Cognos et Edibatec afin de les convertir en des données représentées dans une ontologie OWL. Pour atteindre notre but, nous avons utilisé des outils essentiels pour écrire des codes de programmation en JAVA, des API XML (JAXP, JAXB), et l'API JENA2. Le choix du langage Java s'appuie sur le fait que ce langage fournit une plateforme pour la construction des applications portables.

JAXP (Java API for XML Processing) est une API pour le traitement des fichiers XML. Cette API fournit les deux parseurs SAX (Simple API for XML), et DOC (Document Object Model). Ces deux parseurs permettent d'analyser un fichier XML et d'accéder aux données représentées dans son contenu. Les développeurs de JAVA mettent en application ces deux parseurs par l'intermédiaire de l'API JAXP pour analyser un document XML, le parcourir, et le décomposer en parties discrètes. Le contenu analysé sera disponible pour l'application. Dans l'approche SAX, le parseur commence par le début du document, et passe chaque partie du document à l'application dans la séquence qu'il l'a trouvée. Rien n'est enregistré dans la mémoire. Dans l'approche DOM, le parseur crée un arbre d'objets qui représentent le contenu et l'organisation des données dans le document. Dans ce cas, l'arbre existe en mémoire. L'application peut naviguer dans l'arbre et accéder aux données qu'elle a besoin. JAXB (Java Architecture for XML Binding) est une API qui facilite l'accès aux données XML. Cette API sera détaillée dans la partie suivante.

Jena est une API en RDF qui sert à créer, gérer et manipuler des ontologies. Ainsi, les formalismes d'ontologies qui peuvent être manipulés par cette API sont ceux qui sont construits au dessus de RDF, spécifiquement il s'agit des langages RDFS, DAML+OIL, et OWL et ses trois sous langages. Pour cela, chacun de ces langages a un « *profil* » qui liste les constructions permises, et les URI des classes et des propriétés.

Le projet bcXML a ses instances qui sont décrites dans un document XML, e-Cognos est représenté dans une ontologie DAML+OIL écrite avec le langage RDF/XML, et Edibatec qui a juste des instances présentées par le langage HTML.

II.3.2.1 - bcXML : De XML vers OWL

Pour la ressource bcXML, la structuration des classes est représentée dans un schéma XML, et toutes les instances sont décrites dans des documents XML valides par rapport à ce schéma. La conversion de ces documents en OWL est réalisé par un mécanisme qui consiste à générer un ensemble de classes JAVA qui représentent le schéma XML, et les instances de

bcXML, et ensuite à représenter ces classes Java, et leurs instances dans une ontologie OWL créée avec l'API Jena2.

a) La première étape

La première étape est réalisée par l'API JAXB (Java Architecture for JAVA Binding) qui permet de générer des classes JAVA à partir des schémas XML. Cette technologie fournit aussi des méthodes pour faire les actions de « marshalling » et « unmarshalling »[10]. Ce qui nous intéresse est la deuxième action qui permet d'extraire un ensemble d'objets JAVA à partir d'un document instance XML. L'avantage de JAXB est qu'il fournit un mécanisme rapide et commode pour attacher (bind*)* un schéma XML à une représentation en code JAVA, facilitant l'incorporation des données XML.

Le schéma suivant résume cette étape, en figurant la relation compilation qui sert à généraliser les classes JAVA dérivées du schéma XML. EN plus, nous constatons les actions de « marshalling » et « unmarshalling » qui permettent la conversion entre des objets en JAVA, et les concepts dans un document XML. Ces objets sont des instances des classes générées, et le document XML est supposé valide par rapport à son schéma.

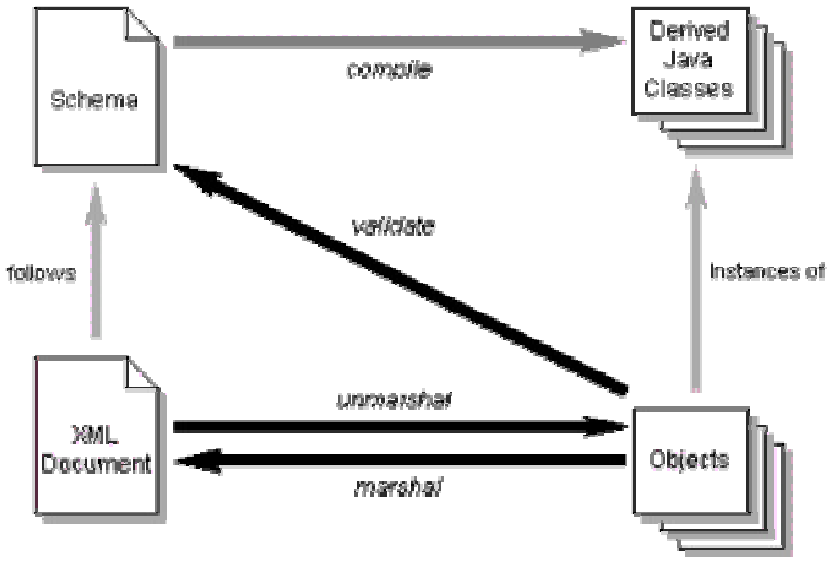


Figure II. 7 : Schéma général de l'API JAXB

Dans la suite, nous allons expliquer les processus qui réalisent le « binding », ou la génération des classes Java à partir du schéma XML.

Le processus de traitement réalisé durant cette étape implique les étapes suivantes :

1) *Binding* : génération des classes d'un schéma XML source, et ensuite compiler les classes générées.
2) *Unmarshalling* : Ce processus génère un ensemble de contenu des objets de données instances des classes du schéma dérivés. Cet arbre de contenu représente la structure et le contenu des documents sources XML conformes à ce schéma.

[10] Le terme « unmarshalling » désigne l'action de convertir un flux d'octets (ici, un document XML) de données en un objet. « marshalling » est le processus inverse qui consiste à convertir un objet en un flux d'octets. Ces deux processus prennent en charge la conversion entre JAVA et XML ;

3) Validation : Le processus « unmarshalling » peut effectuer la validation des documents sources XML par rapport au schéma XML. Dans notre ressource, nous n'avons pas utilisé cette option avant la génération de l'arbre de contenu. Nous supposons que les documents XML de notre ressource sont valides par rapport au schéma du bcXML .

➥ **binding :**

L'application de ce processus sur la taxonomie bcXML permet de générer les classes suivantes :

Les classes types :

DescriptionType : c'est la classe qui permet de décrire une taxonomie, un objet, une propriété, et une valeur restreinte d'une mesure. Elle a les attributs suivants : « Nom », « Explication », « Synonyme », et « référence externe ».

TaxonomyType : C'est le type à laquelle appartient une taxonomie. Elle est une sous-classe de DescriptionType.

ObjectType : Cette classe permet de décrire un objet qui peut exister dans une taxonomie. Cet objet a une propriété, une référence vers le super type et des références sur les parties composantes. Cette classe a une super classe qui est « *DescriptionType* », dont elle hérite tous ses attributs.

PropertyType : C'est une classe qui décrit une propriété d'un objet. Elle peut être soit une mesure, qui a les attributs : « contexte », « unité » et « valeur de restriction », soit une référence à un objet.

ExternalReferenceType : cette classe permet de spécifier une référence externe pour chaque description. Elle est basée sur une chaîne de caractères, et possède un attribut « type » pour spécifier le type de la référence.

TranslatableType : cette classe représente le type des attributs « nom », « synonymes », et « explication » qui sont définis dans la classe « *DescriptionType* ».

Les objets :

Taxonomy : c'est la classe de base de l'exemple bcXML. Elle a un attribut qui spécifie le langage de la taxonomie.

Object : C'est la classe des objets qui est du type ObjectType. Chaque objet a un attribut clé qui permet de l'identifier et le référer à son super objet (objet composite), et à ses objets composants. C'est une sous-classe de « *DescriptionType* », et chaque objet hérite tous ses attributs.

➥ **Unmarshalling :**

« Désagréger » (ou « *Unmarshal* ») un document XML signifie créer un ensemble d'objets qui représente le contenu et l'organisation du document. Ces objets sont des instances des classes produites par le compilateur de « *binding* ».

Pour réaliser ce processus, nous avons écrit un codage de programmation JAVA implémentant les étapes suivantes de l'algorithme de « *unmarhsalling* » (voir annexe) :

- Créer un objet JAXBContext. Cet objet fournit le point d'entrée à l'API JAXB. Pour créer cet objet, il faut spécifier le chemin « Path ». Ce chemin est la liste d'un ou plusieurs noms de packages qui contiennent les interfaces générées par le compilateur de « binding ».
- Créer un objet « Unmarshaller » : cet objet contrôle le processus de « unmarshalling ». En particulier, il contient des méthodes pour exécuter des opérations de « unmarshalling ».
- Appeller la méthode « unmarshal ». Cette méthode effectue le « unmarshalling » du document XML.
- Utiliser les méthodes « *get* » définies dans les classes du schéma pour accéder aux données XML. En fait, les classes générées par le compilateur de « Binding » pour un schéma incluent des méthodes « *get* » et « *set* » qui peuvent être utilisées respectivement pour obtenir et spécifier des données pour chaque instance et attribut du schéma.

L'algorithme I (présenté en annexe) réalise le «*unmarhsalling* » des documents XML du projet bcXML, tout en tenant compte des étapes citées au dessus, et de l'organisation des données selon le schéma du document.

b) La deuxième étape

Cette deuxième étape consiste à créer une ontologie décrite avec le sous langage OWL DL. L'accès au document XML et son analyse nous permettent de réaliser cette étape en se servant de l'API Jena. Cette dernière nous permet de créer une ontologie OWL, ajouter des classes à cette ontologie, et créer des propriétés en précisant son domaine et son « range ». Nous pouvons réaliser la construction d'une ontologie OWL tout en suivant les instructions suivantes :

- Création d'un modèle d'ontologie par l'intermédiaire de la classe ModelFactory. Pour créer un modèle d'ontologie pour le langage OWL DL, on spécifie le profil du langage défini par la variable OWL_DL dans la classe ProfileRegistry :

```
OntModel m = ModelFactory.createOntologyModel (ProfileRegistry.OWL_DL) ;
```

- Créer dans le modèle les classes Java générées lors du processus de «*binding* ». OntClass est la classe qui représente une classe dans un modèle d'ontologie OWL. L'instruction de construction d'une classe est utilisée à chaque fois qu'on accède à une classe JAVA dans l'algorithme de « *unmarshaling* ».

```
OntClass c = m.createClass (JavaClasses);
```

- Pour ajouter dans le modèle une propriété, on utilise la classe OntProperty. Cette classe a un « domain » qui limite les individus à qui s'applique une propriété, et un « range » qui limitent les valeurs que cette propriété peut prendre. En OWL, il existe deux types de propriétés : « DatatypeProperty » et « ObjectProperty ». Le « range » spécifie lesquels de ces deux types est utilisé : le « range » de la première a seulement des données littérales, tandis que celui de la seconde peut également comprendre des instances des classes.

```
ObjectProperty p = m.createObjectproperty (propertyURI) ;
p.addDomain (Class c1)
p.addRange (class c2)
```

- Enfin, durant le processus de « unmarshalling », on accède à des instances ou des objets des classes appartenant à la taxonomie. Pour créer ces instances dans le modèle d'ontologie, on utilise la classe Individual :

```
Individual inst = m.createIndividual (instance, c);
```

II.3.2.2. eCognos : De RDFS vers OWL

Le projet e-Cognos se sert, pour représenter ses données, du langage d'ontologies DAML+OIL, mais n'utilise que l'espace de noms de RDF et RDFS. Il étend RDF et RDFS par des ressources ayant comme type des constructions de l'ontologie DAML+OIL. L'API Jena permet de gérer un document RDFS, de distinguer les « statements » qui y existent, c'est-à-dire la liste des ressources qui jouent le rôle d'un sujet, l'ensemble des propriétés associées à ces sujets (« prédicat »), et les valeurs de ces différentes propriétés (« objets »).

L'algorithme II, présenté en annexe, explique les instructions qui permettent de prendre les ressources du document RDF, ces dernières se limitent à des types classes RDFS, classes DAML+OIL, et des propriétés DAML+OIL, notamment « ObjectProperty » et « DatatypeProperty ». Les propriétés caractérisant ces ressources permettent la hiérarchisation par les relations « rdfs :subClassOf » entre différentes classes, et «rdfs :subPropertyOf » entre propriétés. De plus, on trouve des propriétés qui servent à ajouter un label, et un commentaire à une ressource, en plus de spécifier le domaine et le « range » pour une propriété. Cet algorithme permet de convertir ce document vers une ontologie OWL écrite avec la syntaxe XML/RDF.

Notons que cet algorithme prend en considération les types et les propriétés qui sont inclus dans le document représentant les données de e-Cognos, mais on peut générer un algorithme plus générique qui permet de traiter différentes ressources ayant des types appartenant à des constructions DAML+OIL, et enfin les exprimer par des ressources décrites avec le langage OWL.

II.3.2.3. Edibatec : des fichiers de données vers OWL

Le dictionnaire Edibatec est disponible en deux versions : une version HTML qui permet à l'aide des liens hypertextes de consulter et de visualiser les attributs des différentes classes et une version au format Excel. Ce fichier contient les différentes instances des produits, chacun sur une feuille de calcul composée d'une table ayant comme colonne les caractéristiques des attributs techniques, notamment le code Edibatec, le libellé, le type de saisie, le choix possible, l'unité, et le commentaire du produit. Chaque ligne est un tuple représentant un attribut technique du produit. Dans la table suivante, on a un exemple sur la description de trois attributs techniques d'un produit.

Code	Nom préféré	Unité de mesure	Type de saisie	Valeurs	Définition
B329BAA	Maille	mm	Chaîne(10)	-	Maille du treillis Ex.:100*100
B329BAB	Diamètre du fil	mm	Réel(2,0)	-	Diamètre du fil
B329BAC	Longueur	m	Réel(2,1)	-	Longueur
B329BAD	Largeur	m	Réel(2,1)	-	Largeur

Afin de convertir cette représentation en une ontologie OWL, nous suivons le processus d'obtention de l'ontologie en suivant les étapes suivantes :

- Obtenir les données de la ressource sous format du langage XML, et le schéma de ces données.
- « Binding » du schéma XML pour obtenir les classes en JAVA.
- « unmarshalling » des documents XML pour extraire les objets instances.
- Génération d'une ontologie OWL DL permettant la représentation des classes et des objets instances.

Les trois dernières étapes suivent le même mécanisme de conversion d'un document XML en une ontologie OWL. Les processus de ce mécanisme sont décrits en détail dans la partie implémentation du bcXML.

Pour générer le document XML Schéma de la ressource « Edibatec » et représenter les données du catalogue dans un document XML, nous avons trouvé une astuce qui réalise cette étape via une base de données à partir du fichier excel. Ceci est indiqué dans les étapes suivantes :

- Créer avec « Ms Access 2003 » une base de données pour le modèle Edibatec.
- Importer dans cette base de données les feuilles du fichier XML contenant attributs techniques des différents produits. La première ligne contenant les caractéristiques forme les noms des champs dans la table de base de données.
- On peut exporter cette base de données comportant les différentes tables en un document XML, en même temps que son schéma qui valide le contenu de ces documents.

II.3.3 Intégration dans l'application

Au niveau de l'implémentation, notre approche, en participation avec l'étude sur la contextualisation réalisée par Patrick Hoffmann [68], doit être intégrée dans une architecture qui vise à permettre l'intégration syntaxique et sémantique de ressources sémantiques hétérogènes. Cette application est destinée à être intégrée au serveur d'ontologies e-COSer du CSTB. Une vue générale de cette architecture est présentée dans la page suivante.

Dans cette architecture, chaque module est destiné à faire une tâche bien précise. Le module de conversion d'ontologies génère une ontologie OWL à partir de n'importe quelle ontologie ressource. Le module de liaison entre les ressources sources et celles générées en OWL permet de convertir les URI des concepts sources en URI des concepts OWL correspondants et réciproquement. Le serveur e-COSer gère les requêtes des utilisateurs. Dans cette architecture, trois types de requêtes peuvent exister :

- des requêtes de conversion des ontologies,
- des requête de mise à jour de l'ontologie de correspondances qui permet d'ajouter, de modifier, et de supprimer des relations,
- des requêtes d'appariement, qui sont effectivement chargées de déterminer l'ensemble des concepts en relation avec un concept source (dont l'URI est envoyée en requête par le serveur), et de les classer en fonction de la mesure de la distance sémantique entre ces concepts.

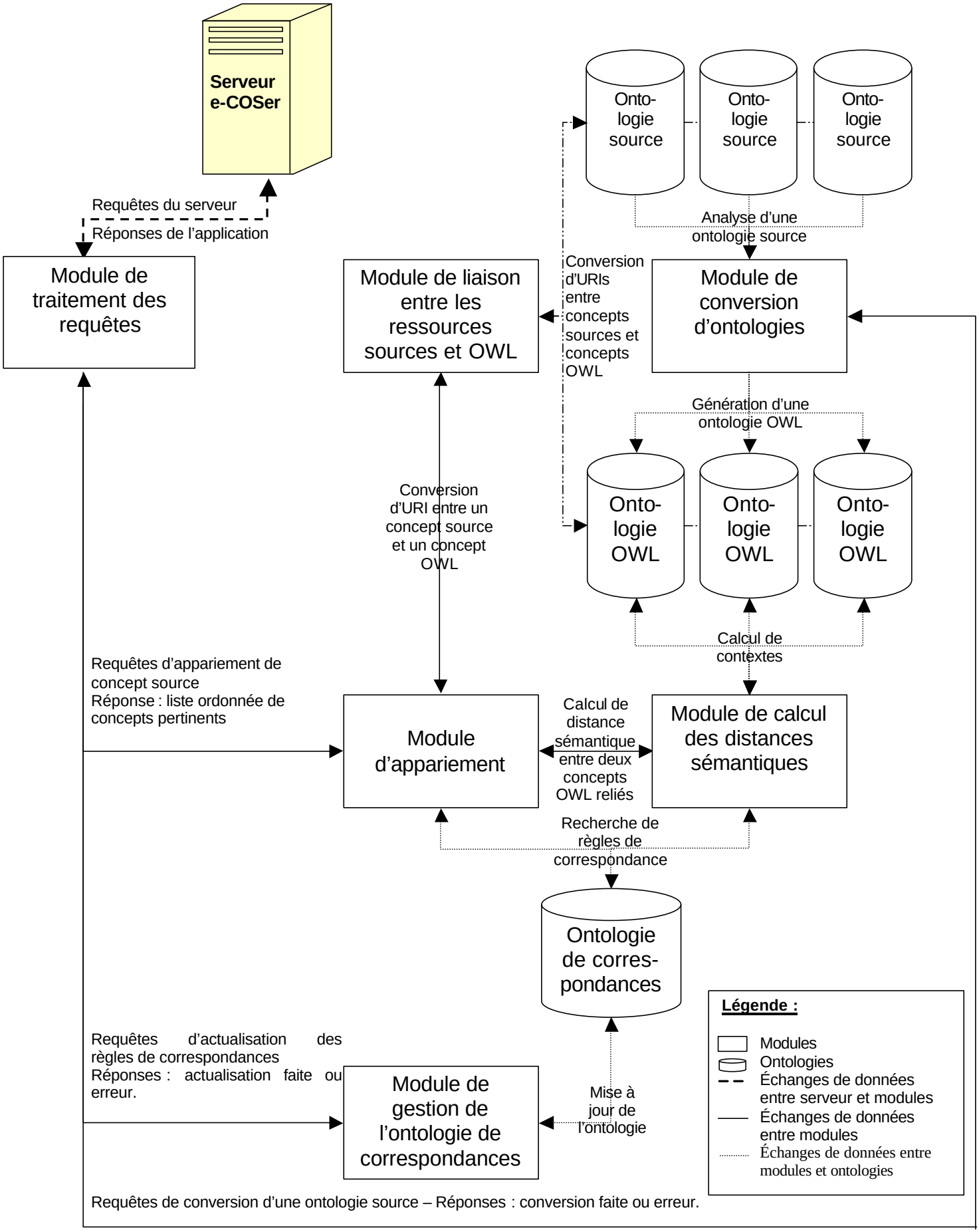


Figure II.8. Schéma général de l'application d'« interopérabilisation » des ressources sémantiques.

CONCLUSION

Dans de nombreux domaines industriels, plusieurs projets de recherches ont abouti à la création d'ontologies hétérogènes, tant au niveau de leurs formats que des données représentées. Afin d'assurer la communication dans le cadre du travail coopératif, ces ontologies doivent être mises en relation afin de pouvoir être intégrées dans les mêmes systèmes. Différentes approches définies dans l'état de l'art ont été proposées pour ce but, et de nombreux problèmes d'hétérogénéité sont analysés.

Dans notre stage, nous avons défini un mécanisme de conversion des langages d'ontologies en OWL. Nous avons expliqué la réalisation de ce mécanisme au niveau des méta-modèles. L'homogénéisation des ontologies sources réalisée au niveau des formats des langages de représentation de la connaissance rend possible la définition des règles de correspondances entre les concepts de différentes sources et entre leurs relations. Ceci est réalisé par la création d'une ontologie qu'on appelle « Ontologie de correspondances ». Cette ontologie s'occupe de la gestion des relations entre ces différentes sources.

Notre approche aide, au niveau générique, à se servir du langage OWL comme d'un langage pivot pour représenter différents types de ressources sémantiques, et explique la conversion de ressources décrites dans d'autres langages (RDFS et DAML+OIL) vers ce langage pivot. Cette unicité de langage est nécessaire pour faciliter la communication et l'interopérabilité des données dans un domaine.

La démarche suivie n'aboutit pas à résoudre complètement le problème de l'interopérabilité sémantique des données. Nous nous limitons dans notre travail à établir des relations entre des concepts, qui peuvent être valides dans certains contextes mais pas dans d'autres. Une prise en compte du contexte et une étude de la sémantique du contenu dans les relations entre concepts est réalisée en parallèle de ce stage par un autre stagiaire en DEA, Patrick Hoffmann, avec lequel nous avons collaboré pour la conception générique de l'application.

Par ailleurs, au niveau de la conversion des langages d'ontologies vers OWL, nous avons considéré des constructions des langages existants dans les documents sources que nous avons convertis. Mais cette approche peut comporter des limites. Notamment, cette conversion doit respecter la sémantique du langage cible (ici OWL DL). Par exemple, nous ne pouvons pas avoir certaines relations d'égalité entre classes et individus dans un langage graphique, car leur conversion en langage OWL DL n'est pas autorisée.

Nous avons appliqué notre approche au domaine de la construction et du bâtiment, en utilisant trois ressources fournies par le Centre Scientifique et Technique du Bâtiment : bcXML, e-Cognos, et Edibatec. Ces trois ressources sont dissemblables en termes de structure, de langage de représentation, et de sémantique des données. Pour cela, nous avons travaillé à partir de modèles spécifiques à chaque projet, soit par l'utilisation de modèles existants, soit par l'élaboration de nouveaux modèles. Ensuite, nous avons défini des algorithmes pour convertir les données des trois ressources, et les représenter dans le langage OWL DL. La conversion en OWL et l'intégration des trois ressources ci-dessus permet à l'utilisateur d'accéder à un ensemble de produits de construction catalogués fournis par le serveur d'ontologies e-COSer « e-Cognos Ontology Server », développé au CSTB.

Ce travail ne se limite pas à cette réalisation de la conversion, et à la définition des règles de correspondances. Des travaux ultérieurs peuvent être menés. En effet, nous n'avons pas pris en compte les contraintes appliquées à des concepts dans une ontologie car nous n'avons pas travaillé avec des ontologies qui en comportaient. Une perspective de ce travail s'oriente donc vers la formalisation des relations pour l'intégration d'ontologies. Dans ce cadre, il s'agit non seulement de réaliser le mapping décrivant l'équivalence sémantique entre deux concepts, mais aussi de considérer des relations plus complexes.

Ces contraintes permettent d'ajouter de la sémantique aux données et d'exploiter l'expressivité du langage OWL d'une manière plus approfondie. Par exemple, un mécanisme qui permet la conversion des contraintes sur des classes en UML, écrites avec OCL, en des contraintes représentées en langage OWL peut être étudié. C'est la raison pour laquelle nous avons choisi OWL DL comme langage pivot. En effet, OWL DL permet d'exprimer des contraintes entre des concepts appartenant à des ontologies différentes et d'effectuer du raisonnement sur ces contraintes. Cela peut avoir de nombreuses applications en conception collaborative, notamment pour l'interopérabilité des modèles de conception, la détection des conflits, etc.

Bibliographie

[1] C. Welty, (2000), "**Towards a Semantics for the Web**", in *Proceedings of Invited presentation at the Dagstuhl Symposium on Semantics for the Web*, Dagstuhl, Germany, May.

[2] N. Guarino, (1997), "**Understanding, Building, And Using Ontologies**", *International Journal of Human Computer Studies. Special Issue on using Explicit Ontologies in KBS Development*, vol. 43 (516): 625-640.

[3] M. N. Huhns and M. Singh, (1997), "**Ontologies for Agents**", *IEEE Internet Computing*, vol. 1: 81-83.

[4] A. Maedche and S. Staab, (2001), "**Comparing Ontologies-Similarity Measures and a Comparison Study**", *Tecnical Report N°Technical Report 408*, Intitute AIFB, University of Karlsruhe, Germany.

[5] Tim Berners-Lee, James Hendler, Ora Lassila, **The Semantic Web**, *Scientific American*, May 2001.

[6] Bowers, S., Delcambre, L.: **Representing and transforming model-based information**. *In: Proc. Int.Workshop on the Semantic Web at the 4th European Conference on Research and Advanced Technology for Digital Libraries (SemWeb)*. (2000)

[7] Michel Biezunski, Martin Bryan, and Steve Newcomb, editors. ISO/IEC 13250, **Topic Maps**, URL:http://www.ornl.gov/sgml/sc34/document/0058.htm.

[8] Michael K. Smith, Chris Welty, and Deborah L. McGuinness, Editors, **OWL Web Ontology Language Guide**,*W3C Recommendation, 10 February 2004*. http://www.w3.org/TR/owl-guide/

[9] Tom Gruber. **A translation approach to portable ontology specifications**. *Knowledge Acquisition*, 5(2):199–220, 1993.

[10] J. Broekstra, M. Klein, S. Decker, D. Fensel, F. van Harmelen, and I. Horrocks. **Enabling Knowledge Representation on the Web by extending RDF Schema**. *In Proceedings of the tenth World Wide Web conference* WWWW'10, Hong Kong, May 2001.

[11] T. Bray, J. Paoli, C.M. Sperberg-McQueen and E. Maler, Editors. **Extensible Markup Language (XML) 1.0, Second Edition**, *World Wide Web Consortium*. 6 October 2000. http://www.w3.org/TR/REC-xml.

[12] Nicolas Routledge, Linda Bird, and Andrew Goodchild: **UML and XML Schema,** *Thirteenth Australasian Database Conference* (ADC2002), Melbourne, Australia. (2001).

[13] H. S. Thompson, D. Beech, M. Maloney, and N.Mendelsohn (Eds). **XML Schema Part 1 : Structures**, W3C Recommendation, May 2001. *http://www.w3.org/TR/xmlschema-1/*.

[14] National Committee for Information Technology Standards, Technical Committee T2 (Information Interchange and Interpretation). **Draft proposed American national standard for Knowledge Interchange Format**. http://logic.stanford.edu/kif/dpans.html, 1998.

[15] A. Farquhar, R. Fikes, and J. Rice. **The Ontolingua Server**: a tool for collaborative ontology construction. In *Proceedings of the 10th Knowledge Acquisition for Knowledge-Based Systems Workshop (KAW'96)*, 1996.

[16] Franz Baader, Diego Calvanese, Deborah McGuinness, Daniele Nardi, Peter Patel-Schneider, editors. **The Description Logic Handbook**. *Cambridge University Press*, 2003;

[17] R. J. Brachman and J. G. Schmolze. **An overview of the KL-ONE knowledge Representation System**. *Cognitive Science*, 9(2):171–216, April 1985.

[18] Deborah L. McGuinness and Frank van Harmelen, Editors. **OWL Web Ontology Language Overview**, *W3C Recommendation*, 10 February 2004. http://www.w3.org/TR/owl-features/.

[19] Dan Brickley and R. V. Guha, Editors. **RDF Vocabulary Description Language 1.0: RDF Schema**, *W3C Recommendation*, 10 February 2004.http://www.w3.org/TR/rdf-schema/

[20] **Dan** Connolly, Frank van Harmelen, Ian Horrocks, Deborah L. McGuinness, Peter F. Patel-Schneider, and Lynn Andrea Stein. **DAML+OIL Reference Description.** *W3C Note* 18 December 2001. `http://www.w3.org/TR/daml+oil-reference`.

[21] Mike Dean and Guus Schreiber, Editors, **OWL Web Ontology Language Reference,** *W3C Recommendation*, 10 February 2004, http://www.w3.org/TR/owl-ref/.

[22] D. Fensel, S. Decker, M. Erdmann, and R. Studer. **Ontobroker: The very high idea**. In *Proceedings of the 11th International Flairs Conference (FLAIRS-98)*, Sanibal Island, Florida, May 1998.

[23] I. Horrocks, D. Fensel, J. Broekstra, S. Decker, M. Erdmann, Goble, F. van Harmelen, M. Klein, S. Staab, R. Studer, and E. Motta. **OIL: The Ontology Inference Layer**. *Technical Report IR-479*, Vrije Universiteit Amsterdam, Faculty of Sciences, Sept. 2000. http://www.ontoknowledge.org/oil/.

[24] S. Cranefield and M. Purvis. **UML as an ontology modelling language**. In *Proceedings of the Workshop on Intelligent Information Integration, 16th International Joint Conference on Artificial Intelligence (IJCAI-99)*, 1999. http://sunsite.informatik.rwth-aachen.de/Publications/CEUR-WS/Vol-23/cranefield-ijcai99-iii.pdf.

[25] G. Booch. **Object-Oriented Analysis and Design with Applications**. *Addison-Wesley, 2nd edition*, 1994.

[26] [Rumbaugh *et al.*, 1998] James Rumbaugh, Ivar Jacobson, and Grady Booch. **The Unified Modeling Language Reference Manual**. *Addison-Wesley*, 1998.

[27] Jos B. Warmer and Anneke G. Kleppe. **The Object Constraint Language: Precise Modeling With UML.** *Addison-Wesley*, 1998.

[28] Cook, S., Daniels, J.: Designing Object Systems: **Object-Oriented Modelling with Syntropy**. Prentice Hall, UK (1994).

[29] D'Souza, D., Wills, A.: **Objects, Components and Frameworks with UML: The Catalysis Approach**. *Addison-Wesley*, to appear 1998. http://www.trireme.com/catalysis

[30] Richard F. Paige and Jonathan S. Ostroff : **A Comparison of the Business Object Notation and the Unified Modeling Language**. *Beyond the Standard. Second International Conference, Fort Collins, CO, USA*, October 28-30. 1999.

[31] B. Meyer. **Eiffel: the language**. *Prentice-Hall*, 1992.

[32] **Klass Objecten for quality in Object and Component Technology** http://www.klasse.nl/ocl/ocl-introduction.html

[33] S. Cranefield and M. Purvis. **Extending agent messaging to enable OO information exchange**. In R. Trappl, editor, *Cybernetics and Systems 2000: Proceedings of the 2nd International Symposium "From Agent Theory to Agent Implementation" (AT2AI-2) at the 5th European Meeting on Cybernetics and Systems Research (EMCSR 2000)*, Vienna, 2000. Austrian Society for Cybernetic Studies. http://www.otago.ac.nz/informationscience/publctns/complete/papers/dp2000-07.pdf.gz.

[34] M. R. Genesereth and N. J. Nilsson. **Logical Foundations of Artificial Intelligence**. Morgan Kaufmann, 1987.

[35] Cranefield S., Haustein S. Purvis M.: **Uml – based ontology modelling for software agents**. *In: Proceedings of the Autonomous Agents 2001 Workshop on Ontologies in Agent Systems* (2001) http://cis.otago.ac.nz/OASWorkshop/Papers/oas01-27-cranefield.pdf

[36] G. Wiederhold, (1994a), "**An Algebra for Ontology Composition**", in *Proceedings of Proceedings of the 1994 Monterey Workshop on Formal Methods*, Monterey, CA, USA, September, pp. 56-61.

[37] P. Mitra, G. Wiederhold, and J. Jannink,(1999a),"**Semi-automatic integration of Knowledge sources**",in *Proceedings of Proceedings of the Fusion'99 Conference*, Sunnyvale, CA, USA, July.

[38] H. Chalupsky, (2000), "**OntoMorph: A Translation System for Symbolic Knowledge**", in *Proceedings of Proceedings of the 7th International Conference on Principles of Knowledge Representation and Reasoning (KR2000)*, Breckenridge, Colorado, USA, April, 12-15, pp. 471-482.

[39] J. Jannink, P. Mitra, E. Neuhold, S. Pichai, R. Studer, and G. Wiederhold, (1999), "**An Algebra for Semantic Interoperation of Semi structured Data**", in *Proceedings of Proceedings of the 1999 IEEE Knowledge and Data Engineering Exchange Workshop (KDEX'99)*, Chicago, USA, November.

[40] F. Hakimpour and A. Geppert, (2001), "**Resolving Semantic Heterogeneity in Schema Integration: an Ontology Based Approach**", in *Proceedings of Proceedings of the*

International Conference on Formal Ontology in Information Systems (FOIS-2001), Ogunquit, Maine, USA, October, 17-19.

[41] Pinto, H. S., G´omez-P´erez, A., and Martins, J. P. (1999). **Some issues on ontology integration**. In *Proceedings of the Workshop on Ontologies and Problem Solving Methods during IJCAI-99*, Stockholm, Sweden.

[42] H. Wache, T. Vgele, U. Visser, H. Stuckenschmidt, G. Schuster, H. Neumann, S. Hbner: "***Ontology-based integration of information A survey of existing approaches"***. In Proceedings of IJCAI-01 Workshop: Ontologies and Information Sharing. 2001.

[43] Yigal Arens, Chun-Nan Hsu, and Craig A. Knoblock. **Query processing in the SIMS information mediator**. In *Advanced Planning Technology*. AAAI Press, California, USA, 1996.

[44] E. Mena, V. Kashyap, A. Sheth, and A. Illarramendi. **Observer: An approach for query processing in global information systems based on interoperability between pre-existing ontologies**. In *Proceedings 1st IFCIS International Conference on Cooperative Information Systems (CoopIS '96)*. Brussels, 1996.

[45] Cheng Hian Goh. **Representing and Reasoning about Semantic Conflicts in Heterogeneous Information Sources**. *Phd, MIT*, 1997.

[46] H. Wache, Th. Scholz, H. Stieghahn, and B. K¨onig-Ries. **An integration method for the specification of rule–oriented mediators**. In Yahiko Kambayashi and Hiroki Takakura, editors, *Proceedings of the International Symposium on Database Applications in Non-Traditional Environments (DANTE'99)*, pages 109–112, Kyoto, Japan, November, 28-30 1999.

[47] Heiner Stuckenschmidt, Holger Wache, Thomas V¨ogele, and Ubbo Visser. **Enabling technologies for interoperability**. In Ubbo Visser and Hardy Pundt, editors, *Workshop on the 14th International Symposium of Computer Science for Environmental Protection*, pages 35–46, Bonn, Germany, 2000. TZI, University of Bremen.

[48] A. Borgida, Brachman a R. J., D. L. McGuiness, and L. A. Resnick**. Classic: A structural data model for objects**. In *ACM SIGMOID International Conference on Management of Data*, Portland, Oregon, USA, 1989.

[49] A.L. Rector, S. Bechofer, C.A. Goble, I. Horrocks, W.A. Nowlan, and W.D. Solomon. **The grail concept modelling language for medical terminology**. *Artificial Intelligence in Medicine*, 9:139 – 171,1997.

[50] Robert M. MacGregor. **Using a description classifier to enhance deductive inference**. In *ProceedingsSeventh IEEE Conference on AI Applications*, pages 141–147, 1991.

[51] D. Fensel, I. Horrocks, F. Van Harmelen, S. Decker, M. Erdmann, and M. Klein. **Oil in a nutshell**. In *12th International Conference on Knowledge Engineering and Knowledge Management EKAW2000*, Juanles-Pins, France, 2000.

[52] François Goasdoué, Véronique Lattes, and Marie-Christine Rousset. **The use of Carin language and algorithms for information integration: The PICSEL project**. *International Journal of Cooperative Information Systems (IJCIS)*, 9(4):383 – 401, 1999.

[53] Diego Calvanese, Giuseppe De Giacomo, and Maurizio Lenzerini. Description logics for information integration. In *Computational Logic: From Logic Programming into the Future (In honour of Bob Kowalski)*, Lecture Notes in Computer Science. Springer- Verlag, 2001.

[54] Vinay K. Chaudhri, Adam Farquhar, Richard Fikes, Peter D. Karp, and James P. Rice. **Open knowledge base connectivity (OKBC) specification document 2.0.3**. *Technical report, SRI International and Stanford University (KSL), April 1998*.

[55] A.D. Preece, K.-J. Hui, W.A. Gray, P. Marti, T.J.M. Bench-Capon, D.M. Jones, and Z. Cui. **The kraft architecture for knowledge fusion and transformation.** In *Proceedings of the 19th SGES International Conference on Knowledge-Based Systems and Applied Artificial Intelligence (ES'99)*. Springer, 1999.

[56] Darrell Woelk and Christine Tomlinson. **The Infosleuth project: intelligent search management via semantic agents**. In *Second World Wide Web Conference '94: Mosaic and the Web*, 1994.

[57] Michael R. Genesereth, Arthur M. Keller, and Oliver Duschka. **Infomaster: An information integration system**. In *1997 ACM SIGMOD Conference.*,1997.

[58] Oscar Corcho and Asuncion Gomez-Perez. **Evaluating knowledge representation and reasoning capabilities of ontology specification languages**. In *Proceedings of the ECAI 2000 Workshop on Applications of Ontologies and Problem-Solving Methods*, Berlin, 2000.

[59] S. Chawathe, H. Garcia-Molina, J. Hammer, K. Ireland, Y. Papakonstantinou, J. Ullman, and J. Widom. **The TSIMMIS project: Integration of heterogeneous information sources**. In *Conference of the Information Processing Society Japan*, pages 7–18, 1994.

[60] Heiner Stuckenschmidt and Holger Wache. **Context modelling and transformation for semantic interoperability**. In *Knowledge Representation Meets Databases (KRDB 2000)*. 2000.

[61] Dieter Fensel, Stefan Decker, M. Erdmann, and Rudi Studer. **Ontobroker: The very high idea**. In *11. International Flairs Conference (FLAIRS-98)*, Sanibal Island, USA, 1998.

[62] Jeff Heflin and James Hendler. Semantic interoperability on the web. In *Extreme Markup Languages 2000*, 2000.

[63] N. Fridman Noy and M. A. Musen, (1999), "**An Algorithm for Merging and Aligning Ontologies: Automation and Tool Support**", in *Proceedings of Proceedings of the Workshop on Ontology Management at the 16th National Conference on Artificial Intelligence (AAAI-99)*, Orlando, FL, USA.

[64] A. Gangemi, D. M. Pisanelli, and G. Steve, "**Ontology Integration: Experience with Medical Terminologies**," in N. Guarino, ed., *Formal Ontology in Information Systems*, IOS Press, 1998, pp. 163-178.

[65] D. McGuinness, R. Fikes, J. Rice, and S. Wilder, (2000a), "**The Chimaera Ontology Environment**", in *Proceedings of Proceedings of the 17th National Conference on Artificial Intelligence (AAAI 2000)*, Austin, Texas, USA, July 30, August 3.

[66] J. Y. Park, J. H. Gennari, and M. A. Musen, (1998),"**Mappings for Reuse in Knowledge - Based Systems**", in *Proceedings of Proceedings of the 11th Workshop on Knowledge Acquisition, Modelling and Management (KAW98)*, Banff, Canada.

[67] N. Fridman Noy and M. A. Musen, (2000), "**PROMPT: Algorithm and Tool for Automated Ontology Merging and Alignment**", in *Proceedings of Proceedings of the 17th National Conference on Artificial Intelligence (AAAI-2000)*, Austin, Texas, USA, July 30, August 3, pp. 450-455.

[68] P. Hoffmann, (2004), "**Appariement contextuel d'ontologies**", *Rapport de stage de DEA*, EDIIS, INSA-Lyon.

ANNEXES

Annexe 1 : *Algorithme d'« Unmarshalling »*

```
Obtenir_Desc_Type (description) ;
//cette procédure permet d'obtenir pour chaque description, soit une taxonomie, un objet, une
// propriété, ou une valeur restreinte la liste en différents langages des noms, des
//explications, des synonymes, et des références externes.
Début
    Avoir_Liste (description .Explications) ;
    Avoir_Liste (description .Noms) ;
    Avoir_Liste (description .Synonymes) ;
    Avoir_Liste (description .Références externes) ;
Fin

Avoir_Liste (paramètre)
Début
//Pour chaque paramètre, on a  les deux procédures Obtenir_langage, et Obtenir_valeur
    Tant que (paramètre a successeur)
    Début
        paramètre.Obtenir_langage ;
        paramètre.Obtenir valeur ;
    Fin
Fin

Procédure_Principale
Début
  //JAXB est une classe qui consiste le point d'entrée à l'API JAXB ; Elle fournit une
  //abstraction pour la gestion d'information de « binding » XML /JAVA nécessaire pour
  // l'implémentation de l'opération « unmarshal »
  JAXBContext jc = JAXBContext.newInstance (packagebcXML) ;

  //Unmarshaller  est une classe dans l'API JAXB qui sert à convertir un document XML en
  // classe JAVA
  Unmarshaller unmarshaller = jc.createUnmarshaller( ) ;
  Taxonomy tax = unmarshaller.unmarshal (document bcXML)

  Langage_taxonomie = tax.Obtenir_langage ;

  //La classe Taxonomy étend la classe DescriptionType,  la conversion de la classe
  //Taxonomy à la classe DescriptionType  nous permet de l'utiliser dans la procédure
  //Obtenir_Desc_Type (Description) afin d'extraire les attributs attachés à la classe
  //DescriptionType.
  DescriptionType taxDesc = (DescriptionType) tax ;
  Obtenir_Desc_Type (taxDesc) ;

  // Obtenir_objets est une procédure définie pour la classe Taxonomy lors du  processus de
  //« binding »
  Liste_Objets = taxonomy.Obtenir_objets ;
  Tant que (Objet a successeur)
    Début //boucle Objet
```

Annexe 1 : *Algorithme d'« Unmarshalling »*

```
//la classe objet comprend la procédure d'obtention de son identificateur Obtenir_id ,
//de réf de l'objet/super Obtenir_ObjetSuperRef, des références des objets composants
//Obtenir_objets_part_ref, et la liste de ses propriétés Obtenir_ptés.
idObjet = objet.Obtenir_id ;
objetSuperType = objet.Obtenir_ObjetSuperRef ;

DescriptionType objetDescription = (DescriptionType) objet ;
Obtenir_Desc_Type (objetDesc) ;

//Liste des objets composants
Liste_ObjetsRéférenceComposants = objet.Obtenir_objets_part_ref ;
Tant que (ObjetsRéférenceComposants  a successeur)
  Début
    Objet_ref_composantID = ObjetRéférenceComosants.Obtenir_ID ;
  Fin

//Liste des propriétés appartenant à chaque objet ;
Liste_Ptés_Objet = objet.Obtenir_ptés ;
Tant que (Propriété a un successeur)
  Début //boucle Propriété
      // Chaque propriété peut être soit une liste de mesure, ou une liste des références
      des objets permis. //Cette classe a la procédure qui renvoie l'ID de la propriété.
      Pté_ID = pté.obtenir_ID ;

      DescriptionType Desc_pté = (DescriptionType) propriété ;
      Obtenir_Desc_Type (Desc_pté) ;

      Liste_mesures = pté.Obtenir_liste_mesures
      Tant que (mesure a successeur)
        Début //boucle mesure
          contexte = mesure.Obtenir_contexte ;
          unité = mesure.Obtenir_Unité ;
          Liste_Valeurs_restreintes = mesure.Obtenir_valeur_restreintes ;
          Tant que (Valeur_Restreinte a successeur)
            Début
                Valeur_restreinte_ID = Valeur_Restreinte.Obtenir_Valeur_ID ;
                Descriptiontype valeur_Desc = (DescriptionType ) Valeur_Restreinte ;
                Obtenir_Desc_Type (objetDesc) ;
            Fin
        Fin //boucle mesure

      Liste_ref_objets_permis = pté. Ref_Objets_Permis ;
      Tant que (Ref_Objet_permis a un successeur)
        Début
          Avoir_Ref_Objet_permis ;
        Fin
    Fin //boucle Propriété
  Fin // boucle Objet
Fin  //Procédure principale
```

Annexe 2 : *Algorithme de conversion de RDFS à OWL*

```
OntProperty fonction_ Obtenir_Pté_OWL (Resource r, OntModel o)
Début
    OntProperty temp = o.Créer_Ont_propriété (r.Obtenir_URI)
    Return temp
Fin

OntClass  fonction_ Obtenir_Classe_OWL (Resource r, OntModel o)
Début
    OntClass  temp = o.Créer_Ont_Classe  (r.Obtenir_URI)
    Return temp
Fin

Procédure Principale Conversion_RDF_OWL
RDFSClassUri = « uri de Classe en RDFS »
DAMLClassURI = « uri de classe en DAML »
OntClass classeOWL ; //Variable qui représente une classe en OWL
OntProperty ptéOWL ; //Variable qui représente une propriété en OWL
Début
    // Création d'un modèle RDF qui  contiendra les Données du projet e-Cognos
    Model modèleRDF = ModelFactory.Création_ modèle ;

    //Création d'une ontologie ayant le profil OWL DL, modèle cible résultat de la conversion.
    OntModel modèleOWL = ModelFactory.Création_Modèle_Ontologie(ProfileOWL) ;
    InputStream  fichier = «Document_ e-Cognos »; // Initialisation du document source

    Si (fichier non trouvé)  Alors afficher « erreur de trouver le fichier » ;
    Sinon
    Début
        //Lecture du fichier source par le modèle RDF
        modèleRDF.Lire(fichier) ;

        //boucle affichant tous les statements existant dans le modèle RDF
        Tant que (stm = modèleRDF.Liste_ Des_Statements a successeur) ;
        Début 1
            Ressource sujet = stm.Obtenir_Sujet ;
            //boucle pour obtenir la liste de  toutes les propriétés de la ressource sujet.
            Tant que  (ptéStm = sujet.Liste_Propriétés a successeur)
            Début 2
                //Obtenir le prédicat et l'objet du statement
                Property pté = ptéStm.Obtenir_Prédicat
                RDFNode objet = pté.Obtenir_Objet ;

                //Vérifier si la ressource est du type classe  RDFS  ou DAML+OIL

                Si (pté.Obtenir_Nom =«type» et (Objet=rdfsClassUri ou (Objet=DAMLClassUri ))
                 Alors classeOWL=modèleOWL.CréerClasse(sujet.ObtenirURI) ;
```

Annexe 2 : *Algorithme de conversion de RDFS à OWL*

```
Sinon
Début
    Si objet = « ObjectProperty » Alors
            ptéOWL = modèleOWL.Créer_PtéObjet(sujet.ObtenirURI);

    Si objet = « DatatypeProperty » Alors
            ptéOWL = modèleOWL.Créer_PtéTypeDonnées (sujet.ObtenirURI);
Fin

Si pté.Obtenir_Nom=«domain» Alors
Début
    //Conversion du nœud RDF en ressource RDF
    Resource dom = (Resource) objet ;
    //Procédure qui retourne la propriété qu'on spécifie son «domain»
    ptéOWL = Obtenir_Pté_OWL (sujet, owlOnt) ;
    ptéOWL.Ajout_Domaine (dom) ;
Fin

Si pté.Obtenir_Nom=«range» Alors
Début
    Resource range = (Resource) objet ;
    ptéOWL = Obtenir_Pté_OWL (sujet, owlOnt) ;
    ptéOWL.Ajout_Range (range) ;
Fin

Si pté.Obtenir_Nom=«subPropertyOf» Alors
Début
    OntProperty sub = (OntProperty) objet ;
    ptéOWL = Obtenir_Pté_OWL (sujet, owlOnt) ;
    ptéOWL.Ajout_SuperPropriété (sub) ;
Fin

Si pté.Obtenir_Nom=«label» Alors
Début
    Literal lit = (Literal) objet ;
    //Procédure qui retource la classe à laqelle on ajoute « label »
    classeOWL = Obtenir_Classe_OWL (sujet, owlOnt) ;
    classeOWL.Ajout_Label(lit) ;
Fin

Début
    Literal comment = (Literal) objet ;
    classeOWL = Obtenir_Classe_OWL (sujet, owlOnt) ;
    classeOWL.Ajout_Comment(comment) ;
Fin
```

Annexe 2 : *Algorithme de conversion de RDFS à OWL*

```
                ***Début***
                    Resource sub = (Resource) objet ;
                    classeOWL = Obtenir_Classe_OWL (sujet, owlOnt) ;
                    classeOWL.Ajout_SuperClasse (sub) ;
                ***Fin***

            ***Fin*** //boucle2
        ***Fin*** //boucle1
    ***Fin*** //sinon
***Fin***
```

Annexe 3 : *Schéma XML des données du projet bcXML*

```
<?xml version="1.0" encoding="utf-8"?>
  <xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema"
             xmlns:src="http://nwalsh.com/xmlns/litprog/fragment"
             xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
             xmlns="http://www.bcXML.org/2003/bcXML"
             targetNamespace="http://www.bcXML.org/2003/bcXML"
             elementFormDefault="qualified">
  <xs:import namespace="http://www.w3.org/XML/1998/namespace"
schemaLocation="http://www.w3.org/2001/xml.xsd"/>
  <xs:element name="Taxonomy" type="TaxonomyType"/>

  <xs:complexType name="TaxonomyType">
          <xs:complexContent>
            <xs:extension base="DescriptionType">
              <xs:sequence>
                <xs:element ref="Object" minOccurs="1"
maxOccurs="unbounded"/>
              </xs:sequence>
              <xs:attribute ref="xml:lang" use="required"/>
            </xs:extension>
          </xs:complexContent>
  </xs:complexType>

  <xs:element name="Object" type="ObjectType">
          <xs:key name="ObjectKey">
            <xs:selector xpath="Object"/>
            <xs:field xpath="@id"/>
          </xs:key>
          <xs:keyref name="SupertypeKeyRef" refer="ObjectKey">
            <xs:selector xpath="Object"/>
            <xs:field xpath="SupertypeRef"/>
          </xs:keyref>
          <xs:keyref name="PartKeyRef" refer="ObjectKey">
            <xs:selector xpath="Object"/>
            <xs:field xpath="PartRef"/>
          </xs:keyref>
          <xs:keyref name="RestrictedValueKeyRef" refer="ObjectKey">
            <xs:selector xpath="Object/Property"/>
            <xs:field xpath="AllowedObjectRef"/>
          </xs:keyref>
  </xs:element>

  <xs:complexType name="ObjectType">
          <xs:complexContent>
            <xs:extension base="DescriptionType">
              <xs:sequence>
                <xs:element name="SupertypeRef" type="xs:NCName"
minOccurs="0"/>
                <xs:element name="Property" type="PropertyType"
minOccurs="0" maxOccurs="unbounded"/>
                <xs:element name="PartRef" type="xs:NCName" minOccurs="0"
maxOccurs="unbounded"/>
              </xs:sequence>
              <xs:attribute name="id" use="required" type="xs:NCName"/>
            </xs:extension>
          </xs:complexContent>
  </xs:complexType>

  <xs:complexType name="PropertyType">
```

```
          <xs:complexContent>
            <xs:extension base="DescriptionType">
              <xs:choice>
                <xs:element name="Measure" minOccurs="0"
maxOccurs="unbounded">
                  <xs:complexType>
          <xs:sequence>
            <xs:element name="Unit" minOccurs="0" maxOccurs="1">
              <xs:simpleType>
                <xs:restriction base="xs:string">
                  <xs:pattern value="[a-zA-Z0-9/*^]+"/>
                </xs:restriction>
              </xs:simpleType>
            </xs:element>
            <xs:element name="Context" type="xs:string" minOccurs="0"
maxOccurs="1"/>
            <xs:element name="RestrictedValue" minOccurs="0"
maxOccurs="unbounded">
              <xs:complexType>
                <xs:complexContent>
                  <xs:extension base="DescriptionType">
                    <xs:attribute name="localid" use="required"
type="xs:NCName"/>
                  </xs:extension>
                </xs:complexContent>
              </xs:complexType>
            </xs:element>
            <xs:any namespace="##other" processContents="lax" minOccurs="0"
maxOccurs="unbounded"/>
          </xs:sequence>
          <xs:anyAttribute namespace="##other" processContents="lax"/>
        </xs:complexType>

                </xs:element>
                <xs:element name="AllowedObjectRef" type="xs:NCName"
minOccurs="0" maxOccurs="unbounded"/>
              </xs:choice>
              <xs:attribute name="localid" use="required"
type="xs:NCName"/>
            </xs:extension>
          </xs:complexContent>
  </xs:complexType>

  <xs:complexType name="DescriptionType" abstract="true">
          <xs:sequence>
            <xs:element name="Name" type="TranslatableType" minOccurs="1"
maxOccurs="unbounded"/>
            <xs:element name="Explanation" type="TranslatableType"
minOccurs="0" maxOccurs="unbounded"/>
            <xs:element name="Synonym" type="TranslatableType"
minOccurs="0" maxOccurs="unbounded"/>
            <xs:element name="ExternalReference"
type="ExternalReferenceType" minOccurs="0" maxOccurs="unbounded"/>
            <xs:any namespace="##other" processContents="lax" minOccurs="0"
maxOccurs="unbounded"/>
          </xs:sequence>
          <xs:anyAttribute namespace="##other" processContents="lax"/>
  </xs:complexType>

  <xs:complexType name="TranslatableType" abstract="true">
          <xs:simpleContent>
```

```
            <xs:extension base="xs:string">
              <xs:attribute ref="xml:lang" use="required"/>
            </xs:extension>
          </xs:simpleContent>
  </xs:complexType>

  <xs:complexType name="ExternalReferenceType" abstract="false">
          <xs:simpleContent>
            <xs:extension base="xs:string">
              <xs:attribute name="type" use="optional"/>
            </xs:extension>
          </xs:simpleContent>
  </xs:complexType>

</xs:schema>
```