BURKINA FASO
UNITE-PROGRES-JUSTICE

MINISTERE DES ENSEIGNEMENTS SECONDAIRE,
SUPERIEUR ET DE LA RECHERCHE SCIENTIFIQUE

UNIVERSITE POLYTECHNIQUE DE BOBO-DIOULASSO

ECOLE SUPERIEURE D'INFORMATIQUE

E.S.I

MEMOIRE DE FIN DE CYCLE

en vue de l'obtention du

DIPLOME D'INGENIEUR DE CONCEPTION EN INFORMATIQUE

THEME : OWLConverter, une approche d'alignement de modèles d'ontologies basée sur la transformation de modèles : Application aux modèles PLIB et OWL.

Présenté par :

Ousmane DIALLO NACANABO

Maître de stage : Dr Mickaël BARON

Directeur de mémoire : Dr Loé SANOU

JUILLET 2010

N° : .....-2010/CICI3

REMERCIEMENTS

Ce mémoire n'aurait pas été possible sans l'aide du Laboratoire d'Informatique Scientifique et Industriel (LISI) qui a bien voulu m'accueillir pour mon stage de fin de cycle d'ingénieur.

Je tiens donc à remercier Monsieur Yamine AIT AMEUR, Directeur du LISI pour m'avoir bien accueilli dans son laboratoire et pour l'expérience enrichissante qu'il m'y a fait vivre.

Je remercie tout particulièrement Mickaël BARON, mon maître de stage, et Stéphane JEAN pour m'avoir guidé tout au long de ce stage, et pour m'avoir tant appris.

Merci à Monsieur Loé SANOU, mon superviseur pour ses conseils et aides multiformes, ainsi qu'à tout le corps enseignant de l'Ecole Supérieure d'informatique (ESI) qui se bat pour notre bonne formation.

Merci à tous ceux qui ont contribué à ma formation depuis mes premiers pas à l'école. Je n'oublie pas mes compagnons de classes, mes collègues de stage, mes amis et tous les membres du LISI pour ces bons moments passés en leur compagnie.

Enfin, un grand merci à ma famille d'avoir toujours été présente à mes côtés.

RESUME

Avec le développement d'Internet et des Intranets, l'échange et l'intégration de données contenues dans les bases de données sont devenus des besoins cruciaux. Au cœur de ces problèmes se situe la nécessité d'expliciter la sémantique des données d'une base de données. En tant que modèle permettant de représenter la sémantique des concepts d'un domaine, les ontologies sont apparues comme une solution possible à ces problèmes.

Cependant pour des raisons variées, il existe plusieurs modèles de représentation d'ontologies. Ainsi, on peut citer le modèle RDF (Ressource Description Framework), le modèle OWL (Web Ontology Language) qui est le standard recommandé par le World Wide Web Consortium (W3C) et le modèle PLIB (Part Library) développé par le Laboratoire d'Informatique Scientifique et Industriel (LISI). Les ontologies et données associées issues d'un modèle d'ontologies ne peuvent être directement exploitables dans un autre modèle. Des solutions sont allées donc dans le sens de permettre d'exploiter les ontologies et les données associées issues de ces différents modèles d'ontologies. C'est ce problème principal que vise à résoudre le travail présenté dans ce rapport qui s'intéresse particulièrement au passage du modèle OWL au modèle PLIB. Nous proposons l'outil ***OWLConverter*** pour permettre la conversion automatique des ontologies et données associées du modèle OWL au modèle PLIB.

ABSTRACT

With the development of the Internet and Intranets, exchange and integration of data in databases have become critical needs. At the heart of these problems is the need to clarify the semantics of data in a database. As a model for representing the semantics of domain concepts, ontologies have emerged as a possible solution to these problems.

However for various reasons, there are several models for representing ontologies. Thus, we can quote the RDF (Resource Description Framework) model, the OWL (Ontology Web Language) model which is the standard recommended by the World Wide Web Consortium (W3C) and the PLIB (Parts Library) model developed by the Laboratory LISI (Laboratoire d'Informatique Scientifique et Industriel). Ontologies and associated data from a model of ontologies cannot be used directly in another template. Solutions are therefore gone in the direction of allowing use ontologies and associated data from these different models of ontologies. It is this main problem aims to solve the work presented in this report which is interested particularly in the passing of OWL model to PLIB model. We propose ***OWLConverter*** tool to allow the automatic conversion of ontologies and associated data from OWL model to PLIB model.

TABLE DES MATIERES

TABLE DES FIGURES

LISTE DES TABLEAUX

GLOSSAIRE

API : Application Programming Interface

BDBO : Base de données à base ontologique

BSU : Basic Semantic Unit

ESI : Ecole Supérieure d'Informatique

IDD : Ingénierie de données

IHM : Interface Homme-Machine

ISO: International standardization Organization

LISI : Laboratoire d'Informatique Scientifique et Industrielle

LDD : Langage de définition de données

LID : Langage d'interrogation de données

LMD : Langage de Manipulation de données

OWL : Ontology Web Langage

PLIB : Parts Library – Norme ISO 13584

RDF : Resource Description Framework

SQL : Structured Query Language

STR : Systèmes Temps Réel

URI: Uniform Resource Identifier

XML : eXtensible Markup Langage

W3C : World Wide Web Consortium

INTRODUCTION

Depuis une quinzaine d'années, le concept d'ontologies imprègne les discussions et les pratiques dans les domaines de l'informatique et de l'ingénierie des connaissances (IC). Les pratiques car on ne se limite plus à parler de l'ontologie mais d'ontologies, désignant par cette double nuance grammaticale des artefacts concrets en lieu et place d'une composante de la philosophie.

Ainsi, le Laboratoire d'Informatique Scientifique et Industrielle (LISI) basé au sein de l'Ecole Nationale Supérieure de Mécanique et d'Aérotechnique (ENSMA) de Poitiers travaille depuis plusieurs années sur le domaine des ontologies (modélisation explicite des informations d'un domaine). Un modèle d'ontologies a notamment été développé et permet l'informatisation de catalogues de composants industriels : le modèle PLIB. Pour assurer la persistance de ces ontologies, le laboratoire a mis en place un modèle de Bases de Données à Base Ontologique (BDBO) nommé OntoDB. Un langage de requêtes appelé OntoQL y a aussi été développé pour permettre l'exploitation de cette BDBO.

A l'instar du modèle PLIB, une ontologie peut être représentée avec plusieurs autres langages (OWL, RDF/RDFS, ...), qu'on appelle aussi modèles d'ontologies. Cette diversité de représentations des ontologies fait apparaître le problème d'échange de données entre systèmes basés sur une même ontologie et utilisant des modèles d'ontologies différents pour représenter ces données et connaissances.

Pour un meilleur échange de données, toutes les organisations devraient utiliser le même modèle d'ontologies. Cette situation est loin d'être atteinte pour des raisons différentes (de représentation, de modélisation, sociales, stratégiques, ...). Il fallait donc penser à une solution pour le partage d'informations entre les systèmes adoptant la même ontologie mais représentée par des modèles différents. Nous nous sommes intéressés à cette problématique.

Le présent rapport est structuré en cinq grands points. Dans le premier point, il sera situé le contexte de ce stage à travers la présentation succincte de la structure d'accueil et de l'équipe dans laquelle nous avons travaillé. Le second point viendra poser les jalons de notre étude à travers les définitions et explications des principaux concepts indispensables à la compréhension de la suite des travaux. Après l'état de l'art, le troisième point consistera à poser clairement le problème ainsi que les objectifs recherchés à travers ce projet. Ensuite, il sera question dans le quatrième point, de concevoir des solutions aux problèmes du point précédant avant d'implémenter à proprement dit ces solutions dans la dernière phase.

I. CONTEXTE DU STAGE

Cette partie présente la structure d'accueil où le stage a été effectué, en l'occurrence le LISI, son organisation et ses axes de recherche.

1.1. Structure d'accueil

Le LISI est un laboratoire de recherche en informatique commun de l'ENSMA et de l'université de Poitiers. Il est reconnu par le ministère de l'enseignement supérieur et de la recherche.

Aussi, le LISI entretient de nombreuses collaborations, nationales et internationales, avec le milieu industriel : Renault, Peugeot Citroën, Toshiba, Siemens, Institut Français du Pétrole, Airbus, etc. Il est étroitement impliqué dans un projet de normalisation internationale. En plus, le laboratoire établit des collaborations académiques avec d'autres laboratoires et universités français et étrangers (Japon, Algérie, Etats Unis, Brésil, Allemagne, Burkina Faso, ...). Mon stage de fin de cycle d'ingénieur s'inscrit dans le cadre de cette collaboration.

Quant à son organisation, le LISI est composé de deux équipes principales : l'équipe des systèmes temps réel (STR) développant des recherches qui s'inscrivent dans la problématique générale de la conception et validation de systèmes temps-réel et embarqués et l'équipe d'ingénierie des données (IDD) qui nous concerne particulièrement et que nous décrirons plus en détail dans la section 1.2.

La Figure 1 ci-dessous présente l'organigramme du LISI dépeint suivant les deux grands axes de recherche dont on parlait plus haut.

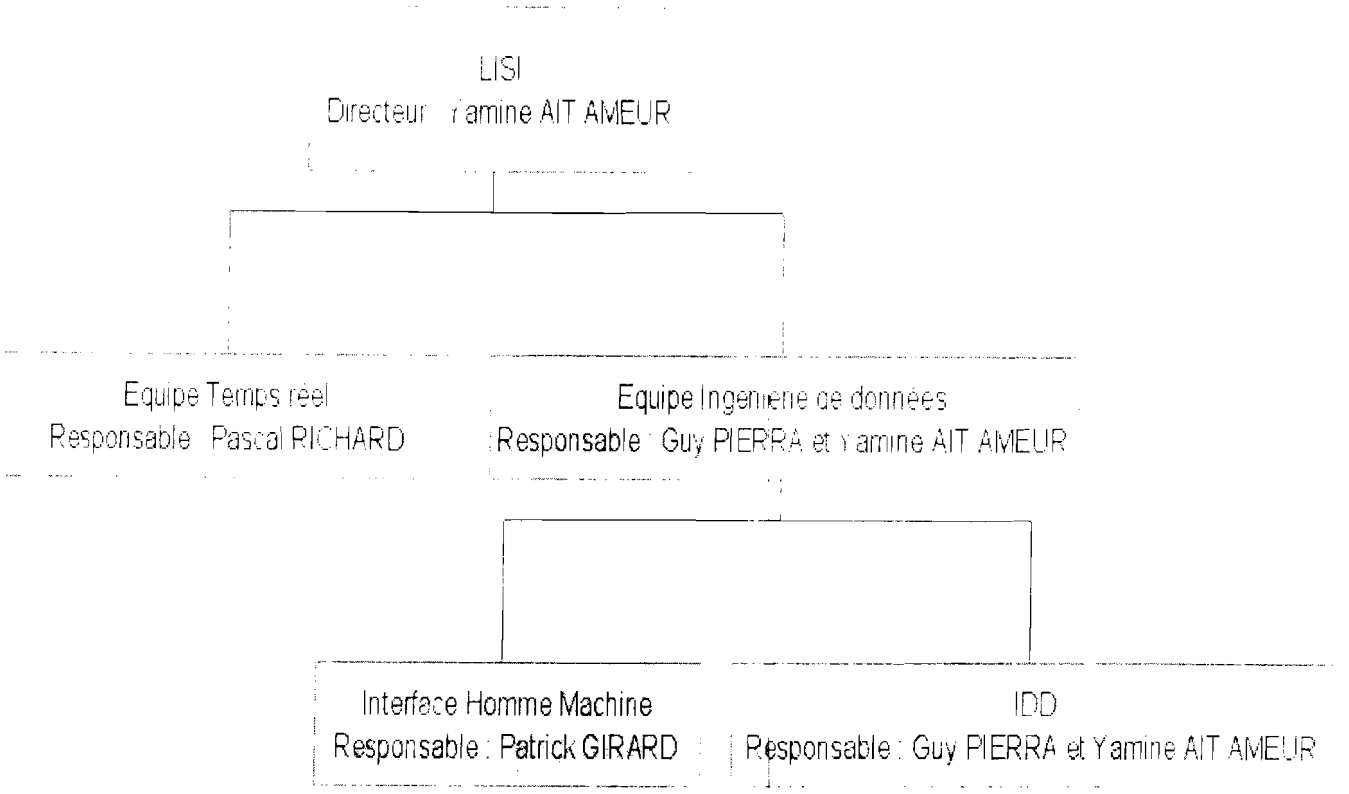

Figure 1 : Organigramme du LISI

1.2. L'équipe ingénierie de données

Les activités de recherche menées dans l'équipe d'IDD s'articulent autour de trois grands thèmes : modélisation à base ontologique, données et interactions et gestion des données persistantes de grande taille.

- **Modélisation à base ontologique**: la notion d'ontologie, concept récent en informatique, correspond à une notion de dictionnaire formel (i.e., traitable par machine) regroupant les concepts existant dans un domaine particulier, par exemple : les instruments de mesure, les composants d'un avion, la géotechnique pétrolière. Un tel dictionnaire peut être référencé à partir des données et programmes, ce qui permet d'en définir la sémantique.

  Les recherches menées autour de cette notion portent sur la représentation des ontologies et sur la modélisation des informations sur un domaine, par référence à une ontologie de ce domaine, ce que l'on appelle modélisation à base ontologique.

  Un modèle d'ontologie original (PLIB), particulièrement adapté au domaine technique, a été développé dans l'équipe et fait l'objet d'un rayonnement international important. Diverses études, à caractère fondamental ou appliqué, portent sur la comparaison de ce modèle avec les autres modèles d'ontologies proposés en particulier dans le cadre du Web sémantique, comme OWL, sur l'intégration automatique de données à base

ontologique, ou sur la mise en œuvre d'une modélisation à base ontologique dans des domaines très variés.

- **Données et Interaction** : Ce thème s'intéresse à l'articulation entre données et interaction. Il traite de la dynamique interactive des données et de leur évolution. Il comporte deux types d'études.

  Le premier type d'études porte sur la représentation sous forme de données du processus constructif ayant généré des ensembles de données. Le principal domaine d'application visé est la géométrie paramétrique qui fait l'objet, depuis quatre ans, d'une collaboration approfondie avec l'équipe MGA (Modélisation Géométrique et Animation) du laboratoire SIC (Signal Image et Communication) de l'université de Poitiers.

  Les autres études portent sur la conception contrôlée d'interfaces homme-machines (IHM) sûres en utilisant les techniques d'ingénierie dirigées par les modèles et la vérification/validation de la qualité des modèles par preuve de propriétés et/ou par le test et l'expérimentation.

- **Gestion des données persistantes de grande taille** : On s'intéresse en particulier aux données à base ontologique qui référencent ou incluent une ontologie. Les travaux menés dans ce cadre portent sur : le développement de nouvelles architectures de bases de données, notamment les entrepôts de données, permettant de gérer des données à base ontologique, la gestion de l'évolution asynchrone des ontologies, l'optimisation physique de leur représentation par fragmentation, indexation et/ou matérialisation de vues.

Nos quatre mois de stage ont été effectués au sein de l'équipe ingénierie des données sous la direction de Yamine AIT AMEUR, Mickael BARON et Loé SANOU.

II. ETAT DE L'ART

2.1. Généralités sur les ontologies

2.1.1. Historique

La notion d'ontologie provient de la philosophie traitant la science de l'être. Cette discipline philosophique, inventée par les grecs, désigne l'étude de l'être et de ses propriétés générales. Le terme « ontologie », emprunté au latin scientifique ontologia, apparait à la fin du XVII éme siècle.

L'ontologie apparait en informatique pendant les années 70 dans le domaine de l'intelligence artificielle. Actuellement, les ontologies sont utilisées dans de nombreux domaines comme l'ingénierie de données, le Web sémantique, la fouille de données, le traitement du langage naturel, l'intégration de l'information, etc.

2.1.2. Définition

Plusieurs définitions ont été proposées pour une ontologie, mais la plus couramment citée reste celle de Gruber : « an explicit specification of a conceptualization » [1]. Dans ses travaux, Gruber a insisté sur le fait qu'une ontologie doit être consensuelle et partageable, choses qui n'ont pas été mentionnées dans sa définition. Jean S, Pierra G et Aït-Ameur Y [2], quant à eux, proposent une définition plus précise, et définit une ontologie comme étant « un dictionnaire formel et consensuel des catégories et propriétés d'entités existant dans un domaine d'étude et des relations qui les lient » [2]. Entité signifie tout élément qui peut exister dans le domaine d'étude. Dictionnaire veut dire que toute entité ou relation dans l'ontologie peut être référencée par un symbole identifiant pour permettre son utilisation dans n'importe quel autre contexte.

2.1.3. Caractéristiques d'une ontologie

Selon Jean S, Pierra G et Aït-Ameur Y, une ontologie est une conceptualisation qui doit représenter les trois caractéristiques suivantes [2] :

- **Formelle** : une ontologie est une conceptualisation basée sur des théories formelles qui permettent de vérifier la consistance et/ou de faire des raisonnements et déductions à partir des ses concepts et de ses instances.
- **Consensuelle** : une ontologie est une conceptualisation acceptée par une communauté. Elle décrit les concepts d'un domaine d'étude pour permettre de satisfaire tous les membres de la communauté. Par exemple, les systèmes développés au sein d'une communauté, et ayant le même domaine d'étude, peuvent être basés sur la même

ontologie. Ceci permet de réaliser des opérations importantes comme l'intégration des systèmes et l'échange de données.

- **Référençable** : chaque concept d'une ontologie est associé à un identifiant unique permettant de le référencer à partir de n'importe quel environnement indépendamment de l'ontologie dans laquelle il a été défini.

2.1.4. Domaines d'utilisation

Les ontologies ont été exploitées pour résoudre de multiples problèmes dans de domaines variés. On peut citer notamment le domaine du traitement du langage naturel, celui de l'ingénierie, de l'interopérabilité des logiciels, du web sémantique et des bases de données.

2.1.4.1. Traitement du langage naturel

Le traitement du langage naturel aborde le problème de la compréhension du langage humain par un ordinateur. Par exemple, il serait intéressant qu'un ordinateur comprenne des questions qu'on lui poserait directement au clavier, telles que : « combien existe-t-il d'employés ayant 15 ans d'ancienneté et gagnant moins de 2000 euros par mois ? ».

Pour résoudre ce genre de problème, une analyse syntaxique et sémantique est indispensable. Ceci conduit à utiliser les ontologies pour construire le lexique utilisé lors de l'analyse syntaxique, et pour traiter les problèmes qui apparaissent lors de l'analyse sémantique tel que la polysémie[1].

Les techniques de traitement du langage naturel sont utilisées dans la recherche d'information. Les moteurs de recherche actuels produisent des résultats en fonction des mots contenus dans les documents parcourus. Des propositions ont été faites pour intégrer les ontologies dans les moteurs de recherche afin de retourner, en plus, des documents pertinents par rapport à la sémantique des mots de la requête [3].

2.1.4.2. Interopérabilité des logiciels

De nos jours, plusieurs logiciels, conçus en utilisant différents modèles, traitent des problèmes identiques sur un même domaine. Donc, un problème d'interopérabilité entre ces logiciels peut apparaitre.

Pour résoudre ce problème, on peut utiliser une ontologie à laquelle seront liés les différents modèles du domaine des logiciels. Cette approche est appelée « Ingénierie logicielle dirigée par les ontologies » [4].

[1] Pluralité de sens d'un mot, d'un signe

2.1.4.3. Web sémantique

Le réseau internet est devenu accessible par tout le monde, et pour des usages différents, tels que le commerce électronique, le passage des examens en ligne, etc. Pourtant, cette technologie souffre d'un défaut majeur. Ce défaut vient du fait que les noms utilisés pour décrire un même service sont différents.

Le but du web sémantique est de développer des ontologies et de catégoriser les services du web selon ces ontologies pour rendre la recherche réalisable d'une manière automatique.

2.1.4.4. Les bases de données

La phase de conception est une étape clé dans la réussite d'une application utilisant une base de données. Et puisqu'une ontologie est une conception sur un domaine d'étude, elle peut être utilisée comme base pour la réalisation de cette phase [5].

L'utilisation des ontologies facilite l'échange de données entre différentes bases de données utilisant la même ontologie. En effet, la signification de chaque élément d'information peut être définie localement en référençant des identifiants d'éléments d'une ontologie.

Par ailleurs, l'intégration de données du même domaine d'étude, provenant de bases de données conçues indépendamment l'une de l'autre pose des problèmes dus à l'hétérogénéité de ces données, notamment les conflits de noms (synonymes et homonymes), les conflits de mesures de valeurs (par exemple, utilisation d'unités de mesure différentes) et les conflits de contextes (concepts semblables mais dans des contextes différents) [6].

Puisqu'une ontologie est un modèle permettant de définir la sémantique des données de différentes sources, elle contribue à fournir une solution pour le problème d'hétérogénéité des données.

2.1.5. Langages de définition des ontologies

La représentation d'une ontologie requiert un langage qui offre les primitives nécessaires pour exprimer les entités et les relations qui existent entre elles ainsi que les opérateurs pour les traiter. Ces langages sont issus de différents domaines comme les bases de données et le web sémantique.

Nous avons étudié le modèle PLIB [ISO 13584-42, 1998, ISO 13584-25, 2004] issu du domaine des bases de données, le modèle RDF/RDFS et le modèle OWL issus du domaine du web sémantique. Nous présenterons dans les paragraphes suivants ces langages de modélisation des ontologies.

2.1.5.1. RDF / RDFS

L'arrivée de XML (eXtensible Markup Language), en 1998, a donné un cadre à la structuration des connaissances, rendant ainsi possible la création de nouveaux langages web destinés non plus à un rendu graphique à l'écran pour un utilisateur humain, mais à un réel partage et à une manipulation des savoirs. C'est dans cet esprit qu'a été créé en 1999 RDF [7] (Resource Description Framework), un langage XML permettant de décrire des métadonnées et facilitant leur traitement.

Le modèle RDF a été conçu initialement par le World Wide Web Consortium (W3C) pour permettre de structurer l'information accessible sur le web et de l'indexer efficacement.

RDF n'est pas particulièrement conçu pour permettre de stocker les métadonnées de documents, mais plutôt pour permettre leur échange et leur traitement par des opérateurs humains ou artificiels. Un des gros avantages de RDF est son extensibilité, à travers l'utilisation des graphes RDF qui peuvent s'intégrer et ne s'excluent pas mutuellement grâce à l'utilisation du concept d'espace de nom « ***namespace*** ».

RDF permet d'exprimer des assertions dont la structure est un triplet (sujet, prédicat, objet). Les éléments du triplet sont définis comme suit :

- Le **sujet** est un URI (Uniform Resource Identifier) identifiant une ressource. Cette dernière peut être une page web ou seulement une partie d'une page web ou toute autre référence à un objet ou concept du domaine étudié.
- Le **prédicat** est une propriété qui caractérise la ressource. Par exemple, une page web peut être caractérisée par le prédicat ***creation_date*** pour définir sa date de création.
- L'**objet** est la valeur de la propriété, elle peut être un URI ou une valeur littérale (texte, nombre, ...).

Ceci peut être illustré par un diagramme composé de nœuds et d'arcs dirigés, dans lequel chaque triplet est représenté par un lien nœud-arc-nœud (d'où le terme de "graphe").

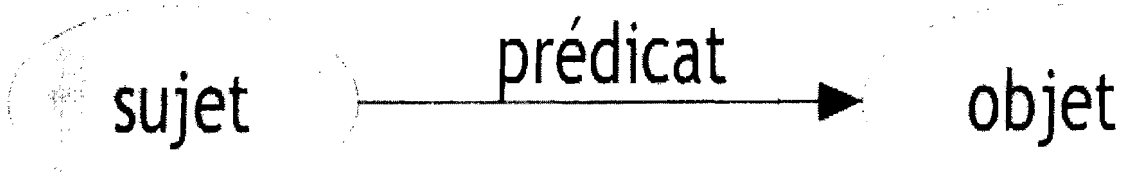

Figure 2 : Le triplet RDF

Dans un graphe, chaque triplet représente l'existence d'une relation entre les choses symbolisées par les nœuds qui sont joints. Le sujet d'un triplet peut être l'objet d'un ou plusieurs autres triplets, ainsi, les triplets RDF peuvent être liés entre eux. L'exemple de la figure 3 ci-dessous illustre deux triplets RDF liés entre eux formant ainsi un graphe RDF.

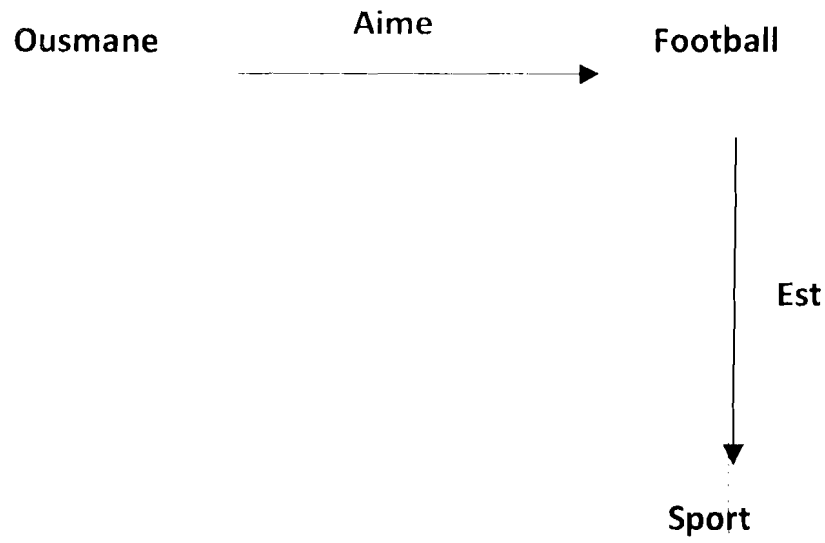

Figure 3 Exemple de graphe RDF

Explication : Dans cet exemple, ***Football*** *est à la fois objet du sujet* ***Ousmane*** *par le prédicat « Aime » et sujet de l'objet* ***Sport*** *par le prédicat « Est ».*

RDFS ou RDF-Schema est une extension du langage RDF, dans lequel plusieurs constructeurs de concepts d'ontologies sont prédéfinis. C'est sur ce modèle qu'est basé OWL (Web Ontology Language) décrit dans le paragraphe suivant.

2.1.5.2. OWL

OWL [8] est, tout comme RDF, un langage profitant de l'universalité syntaxique du XML. Fondé sur la syntaxe de RDF/XML, OWL offre un moyen d'écrire des ontologies web. Il se différencie du couple RDF/RDFS en ceci que, contrairement à RDF, il est justement un langage d'ontologies. SI RDF et RDFS apportent à l'utilisateur la capacité de décrire des classes et des propriétés, OWL intègre, en plus, des outils de comparaison des propriétés et des classes : identité, équivalence, contraire, cardinalité, symétrie, transitivité, disjonction, etc.

Ainsi, OWL offre aux machines une plus grande capacité d'interprétation du contenu web que RDF et RDFS, grâce à un vocabulaire plus large et une vraie sémantique formelle.

En pratique, le langage OWL est conçu comme une extension de RDF et RDFS, il est destiné à la description, par des constructeurs, des classes, des propriétés, etc. De ce fait, il est plus expressif que RDF et RDFS et apporte aussi une meilleure intégration, une évolution, un partage et une inférence plus facile des ontologies.

L'autre avantage, non des moindres, des ontologies OWL réside dans la mise à disposition d'outils capables de raisonner et de faire des déductions automatiques sur elles.

Plus un outil est complet, plus il est, en général, complexe. C'est cet écueil qu'a voulu éviter le groupe de travail ***WebOnt*** du W3C en dotant OWL de trois sous-langages offrant des capacités d'expression croissantes et, naturellement, destinés à des communautés différentes d'utilisateurs :

- ***OWL Lite*** concerne les utilisateurs ayant principalement besoin d'une hiérarchie de classifications et de mécanismes de contraintes simples. Par exemple, même si OWL Lite gère des contraintes de cardinalité, il ne permet que des valeurs de cardinalité[2] de 0 ou 1.

- ***OWL DL*** est plus complexe que OWL Lite, permettant une expressivité bien plus importante. OWL DL est fondé sur la logique descriptive (d'où son nom, OWL Description Logics), un domaine de recherche étudiant la logique, et conférant donc à OWL DL son adaptation au raisonnement automatisé. Malgré sa complexité relative face à OWL Lite, *OWL DL* garantit la complétude des raisonnements (toutes les inférences sont calculables) et leur décidabilité (leur calcul se fait en une durée finie).

[2] Limitation sur le nombre des valeurs prises par une propriété *dans le contexte de cette description de classe particulière*

- ***OWL Full*** est la version la plus complexe d'OWL, mais également celle qui permet le plus haut niveau d'expressivité. OWL Full est destiné aux situations où il est plus important d'avoir un haut niveau de capacité de description, quitte à ne pas pouvoir garantir la complétude et la décidabilité des calculs liés à l'ontologie. OWL Full offre cependant des mécanismes intéressants, comme par exemple la possibilité d'étende le vocabulaire par défaut de OWL.

Il existe entre ces trois sous-langages une dépendance de nature hiérarchique : toute ontologie OWL Lite valide est également une ontologie OWL DL valide, et toute ontologie OWL DL valide est également une ontologie OWL Full valide.

Notre étude a porté sur le sous-langage OWL Lite dont les principaux concepts sont représentés dans la figure 4 :

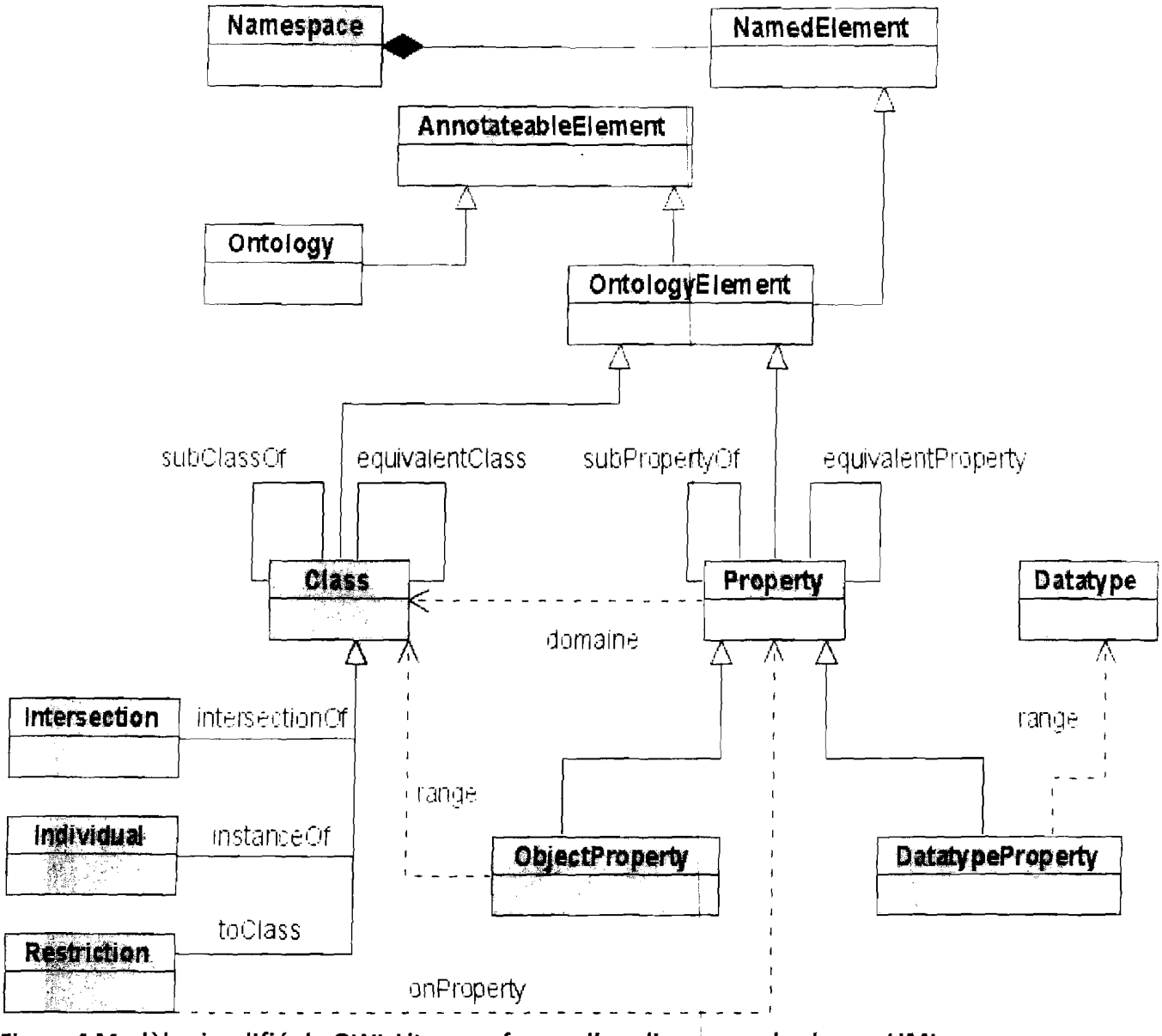

Figure 4 Modèle simplifié de OWL Lite sous forme d'un digramme de classes UML

Nous décrivons dans ce qui suit les constructeurs proposés par OWL Lite.

- ***Constructeur d'ontologies***. ***owl:Ontology*** permet de construire une ontologie OWL. Cette dernière est identifiée par un URI (Uniform Resource Identifier). Une ontologie regroupe des classes et des propriétés. Une ontologie peut être décrite par plusieurs attributs comme ***owl:versionInfo*** qui indique son numéro de version.

- ***Constructeur de classes***. OWL a défini ***owl:Class*** comme constructeur de classes. Chaque classe est identifiée par un URI, et elle est décrite par les attributs textuels du langage RFD/RDFS (***rdfs:label*** et ***rdfs:comment***). Les classes en OWL peuvent être liées entre elles par la relation d'héritage (***rdfs:subClassOf*** de RDFS), qui peut être simple ou multiple

- ***Constructeur de propriétés***. Les constructeurs que OWL a définis pour les propriétés sont ***owl:ObjectProperty*** pour les propriétés dont le codomaine[3] est une instance d'une classe, et ***owl:DatatypeProperty*** pour les propriétés de type simple, c'est-à-dire les propriétés dont le codomaine est un entier, une chaine de caractères, une date, etc. Ces deux constructeurs sont fournis afin de distinguer les propriétés en fonction de leur codomaine. Quel que soit son type, une propriété est définie comme une propriété RDF-Schema ***rdfs:owlProperty***, et caractérisée par un domaine[4] (***rdfs:domain***), par un codomaine (***rdfs:range***).

- ***Constructeur de types de données***. Les types de données d'une propriété que propose OWL sont les types simples définis pour les XML Schéma comme ***xsd:string***, ***xsd:decimal***, ***xsd:int***, etc.

- ***Constructeur d'instances***. ***rdf:type*** est le constructeur défini pour les instances de classes en OWL. Chaque instance est identifiée par un URI, et elle est caractérisée par un ensemble de valeurs de propriétés.

[3] Le type de la propriété pouvant être simple (entier, booléen, ...) ou instance d'une classe

[4] Classe(s) dans la(les)quelle(s) la propriété est définie

2.1.5.3. PLIB

PLIB (Parts LIBrary) est un projet qui a vu le jour en 1987 au niveau européen [ISO13584-42, 1998, ISO13584-25, 2004] et a été élevé au niveau ISO en 1990. Son objectif était de permettre la modélisation de la connaissance en ingénierie, en particulier la modélisation informatique des catalogues de composants.

La figure 5 ci-dessous représente les concepts élémentaires du modèle PLIB. Pour plus de détail sur ces concepts, voir la figure 6 du noyau OntoQL.

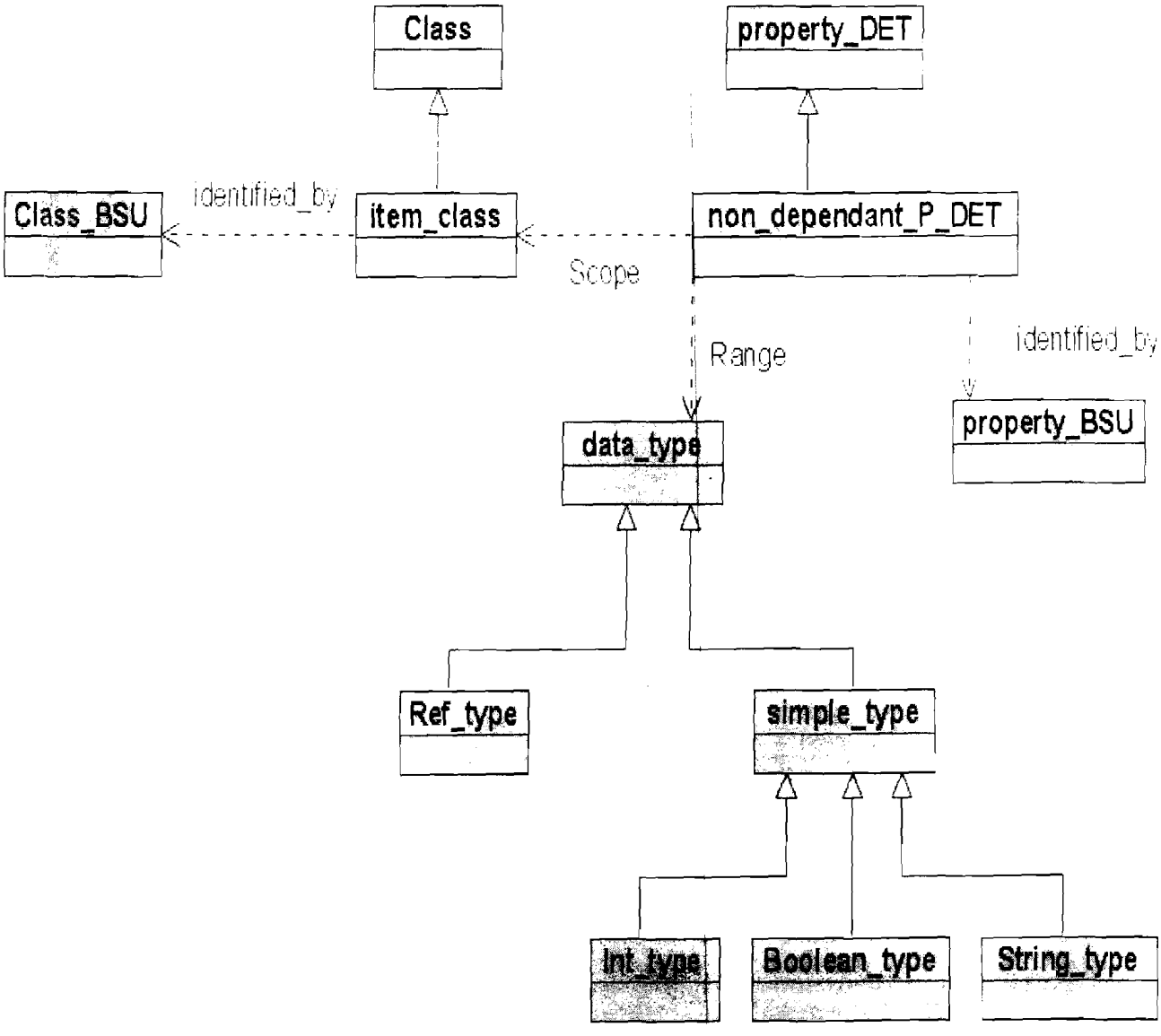

Figure 5 Représentation simplifiée du model PLIB

Le modèle PLIB permet ainsi de créer des ontologies avec les constructeurs suivants :

- ***Constructeur de classes***. ***item_class*** permet de définir des classes en PLIB. A l'image des URI en RDF, ces classes sont identifiées par un identifiant unique appelé **BSU** (Basic Semantic Unit), et décrites par des attributs tels que nom, noms synonymes, définition, note, remarque, version, révision,... dans plusieurs langues. L'opérateur ***is_a*** définit l'héritage simple en PLIB, qui autorise, en particulier, une classe à hériter de toutes les propriétés d'une autre classe. D'autre part, l'opérateur ***is_case_of***, permet à une classe d'importer des propriétés d'une ou plusieurs classes. Cette relation définit une relation d'héritage partiel. Notons que l'héritage multiple n'est pas permis, seul l'héritage simple est autorisé. Il est simulé par la relation is_case_of.

- ***Constructeur de propriétés***. ***non_dependant_p_det*** définit les propriétés en PLIB. Ces dernières sont identifiées par un identifiant BSU et décrites par les mêmes attributs qui décrivent les classes. Le champ d'application d'une propriété est défini par son domaine (***scope***). Son domaine de valeurs est défini par son codomaine (***range***). Celui-ci peut être de type simple (entier, chaine de caractères, date, ...), ou objet, c'est-à-dire une instance d'une classe.

- ***Constructeur de types de données***. Le modèle PLIB propose des constructeurs pour plusieurs types de données primitifs tels que les entiers (***int_type***) ou les booléens (***boolean_type***). Ces derniers peuvent être associés à une unité de mesure (***int_measure_type***) ou à une monnaie (***currency_type***). Pour les types de données spécifiant des instances de classes, PLIB a introduit le constructeur ***class_instance_type*** qui permet d'utiliser une classe comme un type de données. PLIB fournit également le type ***aggregate_type*** pour pouvoir créer des collections qui peuvent être de type simple ou objet.

- ***Constructeur d'instances***. Une instance PLIB est définie par sa classe de base et les valeurs de ses propriétés. Elle appartient à sa classe de base ainsi que les classes mères de sa classe de base. Une instance d'une classe ne fournit pas nécessairement de valeur à toutes les propriétés de la classe. Les valeurs caractérisant les instances de classes peuvent être associées à des unités de mesure ou des paramètres influant sur leur évaluation.

Après avoir défini une ontologie, ses caractéristiques, ses domaines d'utilisation ainsi que les langages pour la définir, nous exposons dans la section suivante les bases de données à base ontologique.

2.2. Les bases de données à base ontologique (BDBO) : Cas de OntoDB

De nos jours, les systèmes de stockage d'information tels que les bases de données sont extrêmement développés et sont de plus en plus volumineux. Cependant, ils possèdent une faiblesse : les données manipulées par les différentes applications n'ont de sens qu'à travers elles et l'utilisation qu'elles en font. Si une donnée est prise isolée de tout programme, il est impossible d'en tirer quelque information que ce soit. L'idée est alors d'associer aux données une sémantique. Ainsi, à chaque donnée est associée une description, un type, une unité de mesure, ou quelque information que ce soit. Le modèle de représentation utilisé pour relier donnée et sémantique est l'ontologie.

Des propositions sont allées vers la sauvegarde des ontologies et leurs données dans des bases de données afin d'assurer leur persistance. D'où la naissance des BDBO.

Une base de données à base ontologique (BDBO) est une source de données qui contient des données et des ontologies qui en définissent la sémantique. Les données stockées dans une BDBO sont appelées des données à base ontologique.

Une BDBO a vu le jour au sein du LISI, nommée ***OntoDB***. OntoDB est implanté sur le Système de Gestion de Base de Données (SGBD) ***PostgreSQL***.

2.2.1. Architecture de OntoDB

L'architecture de OntoDB se décompose en quatre parties, comme l'indique la figure 6 ci-dessous :

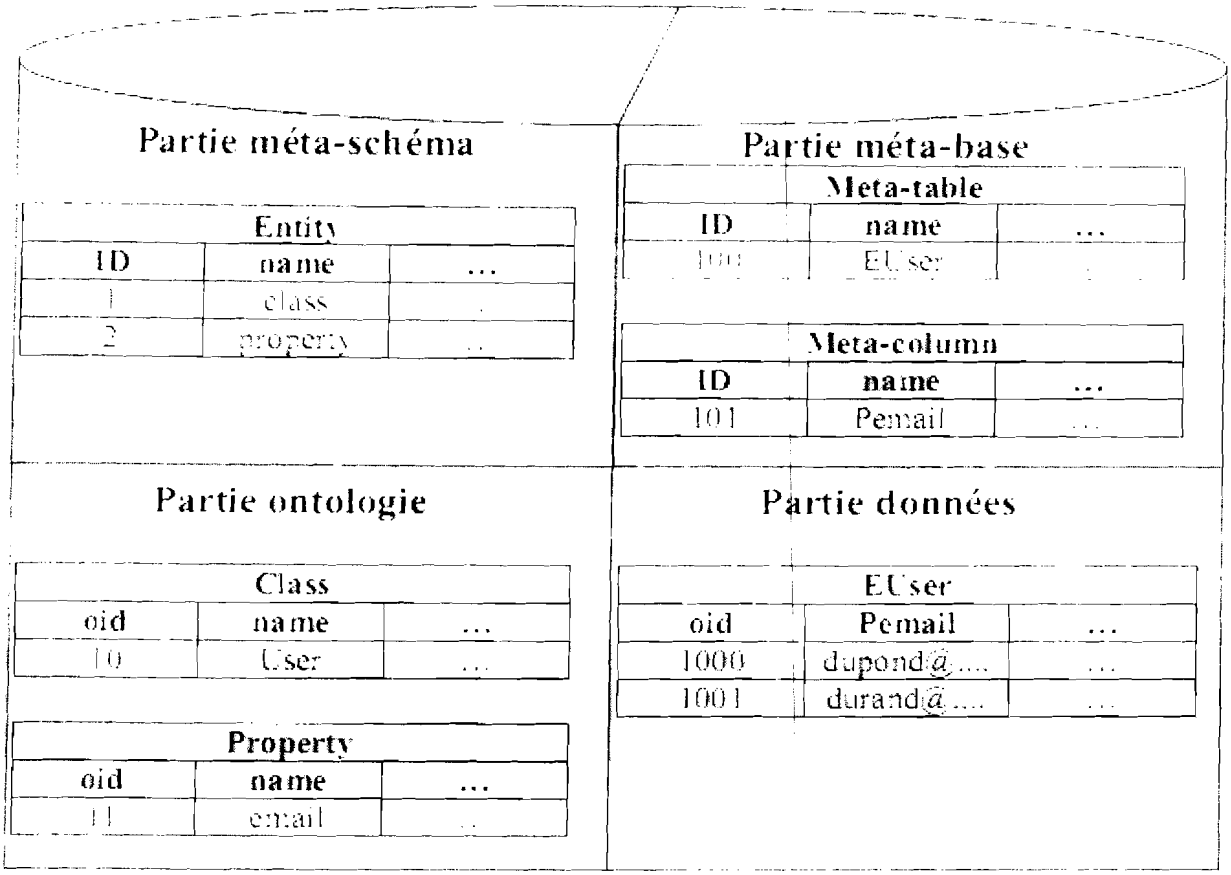

Figure 6 : Architecture OntoDB

- ***La partie métabase*** est la partie traditionnelle des bases de données classiques qui contient les tables décrivant les tables des autres tables de la base. Dans OntoDB, elle contient la description de toutes les tables des trois autres parties.
- ***La partie données*** contient les instances des classes de l'ontologie et les valeurs de propriétés associées à ces instances.
- ***La partie ontologie*** contient les concepts d'ontologies PLIB, le modèle d'ontologies supporté par OntoDB, en l'occurrence des classes et propriétés et les relations entre ces concepts.
- ***La partie méta-schéma*** représente le modèle d'ontologies utilisé et le méta-schéma en même temps. On y trouve, particulièrement dans OntoDB, la description de la notion de classes et de propriétés caractérisant le modèle PLIB. Dans l'exemple, on trouve par exemple les entités « *Class* » et « *Property* ».

2.2.2. Représentation des ontologies

Les ontologies sont sauvegardées, dans OntoDB, au niveau de la partie ontologie, dans des tables suivantes :

- La table ***Ontology*** contient les ontologies stockées dans OntoDB, et elle est composée des colonnes décrites dans le tableau 1.

Tableau 1 : Description des colonnes de la table Ontology

Colonne	Description
oid	l'identifiant de l'ontologie dans OntoDB
namespace	son espace de nom

- La table ***Class*** comporte les classes d'ontologies. Nous présentons dans le tableau 2, quelques attributs permettant de décrire ces classes.

Tableau 2 : Description des colonnes de la table Class

Colonne	Description
oid	l'identifiant de la classe dans OntoDB
code	code de la classe
name[en]	nom de la classe en anglais
name[fr]	nom de la classe en français
definition[en]	définition de la classe en anglais
definition[fr]	définition de la classe en français
shortName[en]	nom abrégé de la classe en anglais
shortName[fr]	nom abrégé de la classe en français
definedBy	l'oid de l'ontologie dans laquelle la classe est définie
usedProperties	les oids des propriétés de la classe
directSuperClasses	les oids des super classes directes

- La table ***Property*** comprend les propriétés des ontologies. Chaque propriété est décrite par les attributs que nous présentons non exhaustivement dans le tableau 3.

Tableau 3 : Description des colonnes de la table Property

Colonne	Description
oid	identifiant de la propriété dans OntoDB
code	code de la propriété
name[en]	nom de la propriété en anglais
name[fr]	nom de la propriété en français
definition[en]	définition de la propriété en anglais
definition[fr]	définition de la propriété en français
shortName[en]	nom abrégé de la propriété en anglais
shortName[fr]	nom abrégé de la propriété en français
scope	l'oid de la classe indiquant le domaine
range	l'oid du type de données représentant le codomaine

2.2.3. Représentation des données à base ontologiques

OntoDB utilise la représentation *horizontale*[5] pour représenter les instances de classes et les valeurs de propriétés associées à ces instances.

Chaque classe définie dans la partie ontologie d'OntoDB possède une table dans la partie données décrivant son extension. Cette table contient une colonne *oid* pour identifier les instances, et une colonne pour chaque propriété portant le nom de cette dernière.

Par exemple si nous considérons une classe *Wine* ayant deux propriétés : *isRed* et *alcoholPercentage*, on aura la représentation suivante :

Tableau 4 : Exemple de représentation des données dans OntoDB

oid	**isRed**	**alcoholPercentage**
001	{115}	4

[5] Comme dans le modèle relationnel

2.3. Un langage d'exploitation des BDBO : OntoQL

OntoQL [9] est un langage, associé à OntoDB et au modèle PLIB, qui permet d'interroger les données et les ontologies. Ce langage est indépendant du modèle d'ontologies utilisé par la BDBO. En effet, ce langage est basé sur un noyau (voir figure 7) commun aux différents modèles d'ontologies et les instructions de ce langage permettent de l'étendre.

Le langage OntoQL permet d'exprimer des instructions dans différentes langues naturelles, ainsi, ce langage est compatible avec le langage SQL ; ce qui lui permet donc d'exploiter les données au niveau logique d'une BDBO. En plus, OntoQL étend le langage SQL pour permettre d'accéder aux données du niveau ontologique indépendamment de la représentation logique des données tout en permettant d'en manipuler la structure.

La figure 7 ci-dessous présente le modèle d'ontologie noyau de OntoQL sous la forme simplifiée d'un diagramme de classe UML :

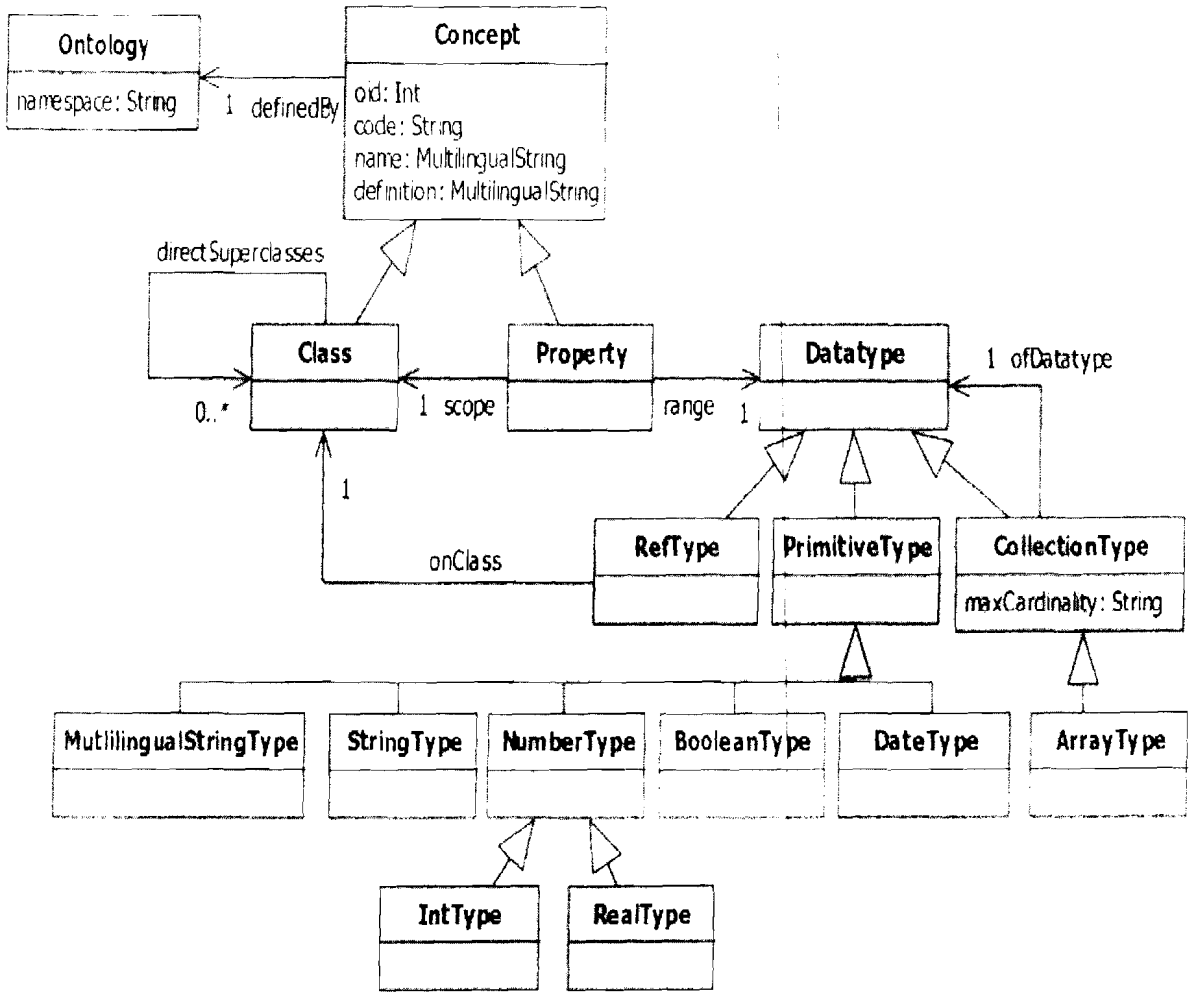

Figure 7 : Le modèle d'ontologie noyau du langage OntoQL

Les éléments de ce modèle peuvent être décrits comme suit :

- Une ontologie (***Ontology***) définit un domaine d'unicité des noms (***namespace***). Elle regroupe la définition d'un ensemble de concepts : des classes et des propriétés.

- Une classe (***Class***) possède un identifiant interne à la BDBO (***oid***) et un identifiant indépendant de celle-ci (*code*). Sa définition comporte une partie textuelle (***name, definition***), éventuellement donnée dans plusieurs langues naturelles (*MultilingualString*). Ces classes sont organisées selon une structure de graphes acycliques qui représente les relations d'héritage (*directSuperclasses*).

- Les propriétés (***Property***) possèdent également un identifiant (interne et externe) et une partie textuelle éventuellement définie dans plusieurs langues naturelles. Chacune des propriétés doit être définie sur la classe ou sur l'une des superclasses des instances qu'elle décrit (scope). Chaque propriété possède un codomaine (range) qui permet de contraindre son domaine de valeurs.

- Le type de données (***Datatype***) d'une propriété peut être un type simple *(PrimitiveType)* tel que le type entier (*IntType*), le type chaîne de caractères (StringType) ou encore un type booléen (*BooleanType*). Une propriété peut également prendre ses valeurs dans une classe en faisant référence à ses instances (*RefType*). Enfin, cette valeur peut aussi être une collection dont les éléments sont soit de type simple soit des références à des instances d'une classe (*CollectionType*).

A partir de ce modèle d'ontologie noyau, on peut définir différents langages de manipulation des données et des ontologies, notamment le langage de définition de données (DDL, Data Definition Language), le langage de manipulation de données (DML, Data Manipulation Language) et le langage d'interrogation de données (DQL, Data Query Language) que nous présentons succinctement ici :

- Le DDL

Le DDL permet de créer de nouvelles classes et propriétés au sein d'une ontologie.

La syntaxe de création d'une classe et de ses propriétés est :

CREATE <entity id> <class id> [<under clause>] [<description clause>] [<properties clause list>]

<under clause> ::= ***UNDER <class id list>***

<description clause> ::= ***DESCRIPTOR (<attribute value list>)***

<attribute value> ::= ***<attribute id>=<value expression>***

<properties clause> ::= ***<entity id> (<property definition list>)***

<property definition> ::= ***<prop id> <datatype> [<description clause>]***

L'élément *<entity id>* spécifie le type de concept à créer, dans notre cas c'est soit ***#Class*** ou ***#Property***, qui sont des entités prédéfinies, et qui servent respectivement à créer les classes et les propriétés d'ontologies. *<class id>* représente l'identifiant de la classe créée, tandis que *<class id list>* spécifie la liste des identifiants des super classes éventuelles après le mot UNDER.

La clause *DESCRIPTOR* est optionnelle. Elle est utilisée pour décrire les classes et les propriétés créées en donnant des valeurs aux attributs de classes *<attribut id>*. *<properties clause>* permet de créer des propriétés de la classe (*<prop id>*) au moment de la création de cette dernière. Les propriétés sont suivies par leurs types de données (*<datatype>*) et éventuellement une nouvelle clause *DESCRIPTOR*.

Exemple :

CREATE #Class Wine

UNDER Thing (DESCRIPTOR (#name [en] = 'Wine')

PROPERTIES (URI String

DESCRIPTOR (#version='01')))

La classe Wine est définie comme une classe (#Class) ayant Thing comme une superclasse. Elle est décrite par son nom en anglais wine et possède la propriété URI de type chaine de caractères (String). Cette propriété est décrite par l'attribut version qui prend la valeur 01.

- **Le DML**

Le DML doit permettre de créer, mettre à jour et supprimer les instances des classes dans une BDBO. Ces instances possèdent un identifiant et ne peuvent présenter de valeurs que pour les propriétés de l'extension dans laquelle elles sont stockées. En conséquence, la syntaxe du langage de manipulation de données reste conforme à celle de SQL.

Cependant, pour manipuler ces instances, une extension doit tout d'abord être définie sur une classe. L'extension est créée dans la partie donnée d'OntoDB et correspond à une table du modèle relationnel dont les colonnes représentent les propriétés valuées des classes.

La syntaxe de création de l'extension d'une classe est la suivante :

CREATE EXTENT OF <class id> (<property id list>).

Exemple : ***CREATE EXTENT OF Wine (URI)***

L'extension de la classe Wine est créée en indiquant que la propriété URI permet de décrire les instances. La table d'extension ne comporte qu'une colonne.

Une fois ces extensions créées, nous pouvons utiliser le DML pour insérer de nouvelles instances :

INSERT INTO Wine (URI)

VALUES ("http://www.ensma.lisi.fr/2010/10/OntologyTest#Wine")

Une instance de la classe Wine est créée en donnant une valeur à la propriété URI. Cela correspond à l'insertion d'une nouvelle ligne dans la table d'extension.

- ***Le DQL***

Dans OntoQL, les DDL et DML sont des extensions du langage SQL, qui répondait naturellement aux besoins : création de classes, d'instances, sélection de données, etc. A l'inverse, le DQL a dû être adapté pour répondre aux exigences spécifiques d'OntoQL. Par exemple, considérons la requête suivante :

SELECT #name FROM #Class WHERE #oid = 1029 (en supposant que 1029 est bien l'identifiant interne de la classe Wine)

Si le langage de travail choisi par l'utilisateur est l'anglais, celle-ci retournera par exemple *'wine'*. Par contre, si le langage de l'utilisateur est le français, la requête retournera *'Vin'*.

Exemple:

SELECT URI from Wine

Using namespace 'http: //lisi.ensma.fr/'

Cette requête permet de sélectionner l'URI de toutes les instances de la classe Wine dépendant de l'ontologie portant l'espace de nom (namespace) http://lisi.ensma.fr/.

III. PROBLEMATIQUE

Il existe plusieurs langages de définition ou modèles de représentations d'ontologies. Le LISI à un modèle qui a été conçu en son sein pour répondre à des besoins spécifiques. Ce modèle, basé sur la représentation PLIB (Part Library) décrite plus haut, utilise comme BDBO OntoDB et OntoQL pour l'exploitation de cette dernière. Cependant, ce modèle à une représentation d'ontologies différente de celle d'OWL (Web Ontology Langage) considéré comme le standard dans le domaine du Web sémantique.

Les ontologies écrites en PLIB et celles écrites en OWL ont des caractéristiques et des contraintes différentes. Dans les ontologies OWL, par exemple, l'héritage multiple est autorisé, tandis que le modèle PLIB n'autorise que l'héritage simple et simule l'héritage multiple.

Notre travail consistera à permettre un échange de données entre les modèles d'ontologies OWL et PLIB.

D'emblée, il nous sera indispensable d'étudier en détail les deux modèles d'ontologies OWL et PLIB, de comprendre les règles de passage d'un modèle à l'autre, en particulier de savoir ce que devient chaque concept lors de sa transformation d'un modèle à l'autre, puis implémenter ces règles en développant une API (Application Programming Interface) Java qui fera la transformation automatique des données du modèle OWL en PLIB.

Aussi, pour pouvoir interagir avec les ontologies dans des programmes informatiques, plusieurs API ont été développées. Chaque API propose un certain nombre de fonctionnalités (interfaces et classes) facilitant l'exploitation des concepts et données à base ontologique. Toute fois, ces API, open source, sont assez vastes et complexes pour une exploitation entière de celles-ci dans la résolution du problème principale qui est la conversion de types. Ainsi le groupe de pilotage du projet a trouvé judicieux de construire, en amont, une API propre au laboratoire, basée sur les deux APIs Java les plus utilisées (JENA et OWL API) en n'extrayant de ces dernières que les fonctionnalités dont notre application finale ***OWLConverter*** aura besoin.

La figure 8 synthétise l'objectif de l'API à développer.

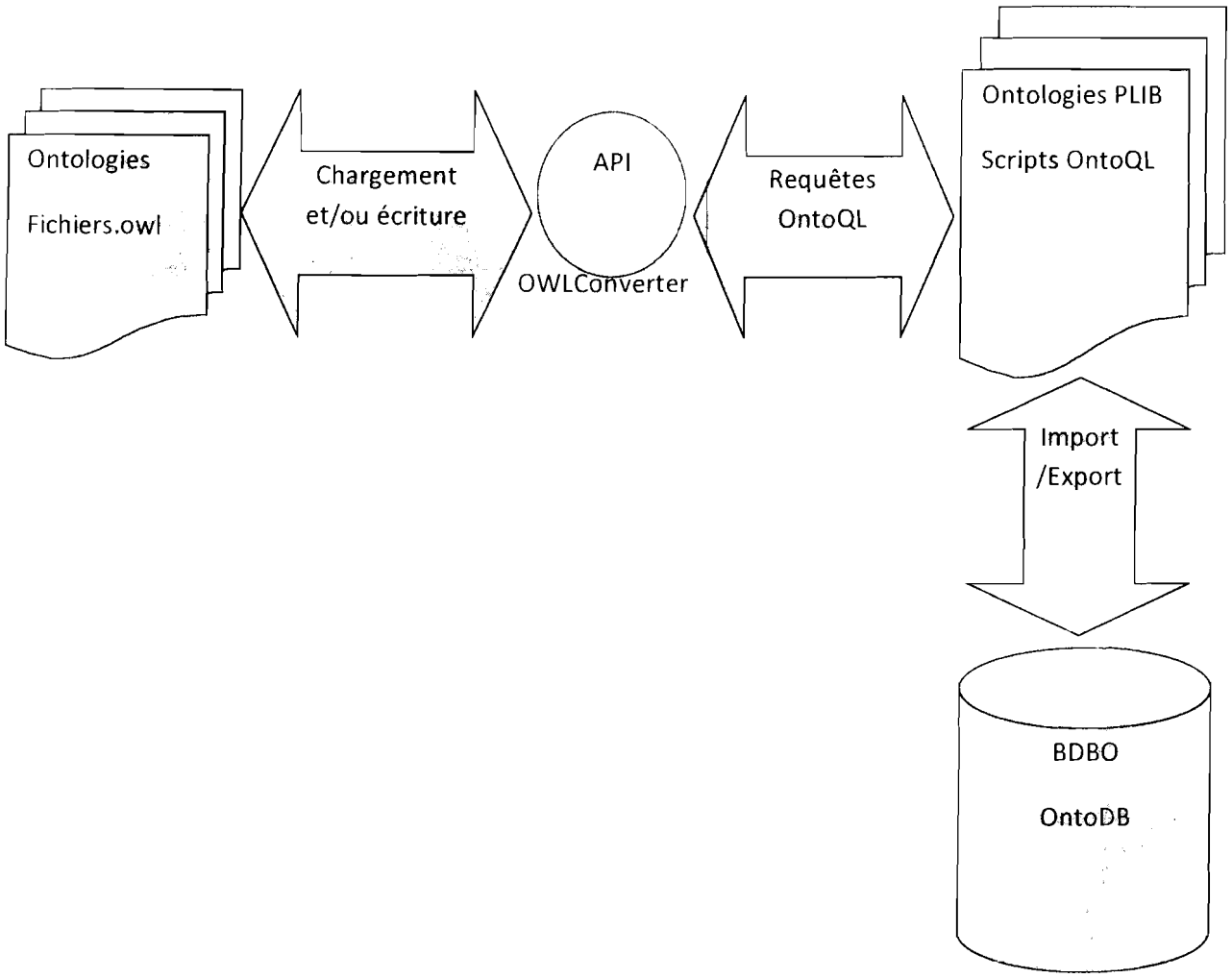

Figure 8 : Vue générale du travail demandé

Explication : D'une ontologie représentée en modèle OWL, OWLConverter doit permettre de charger, parser, extraire les données et concepts pour ensuite appliquer les règles de transformations et obtenir des scripts OntoQL sous forme de requêtes (éventuellement dans un fichier). OWLConverter doit enfin gérer la connexion à la BDBO OntoDB pour y créer l'ontologie PLIB correspondantes avec ses données à partir des requêtes OntoQL.

IV. CONCEPTION

4.1. Les modèles existants

Après avoir posé le problème et montré l'objectif à atteindre à travers ce projet, nous abordons dans cette phase la structuration d'une solution à travers d'abord une étude des modèles d'APIs existants sur lesquels va se baser notre modèle à mettre en place. Ensuite nous verrons les règles de transformation des ontologies OWL en PLIB.

4.1.1. Modèle Jena

Jena[6] est un framework open-source développé par les laboratoires HP (HP Labs Semantic Web Programme). Il est formé d'un ensemble d'outils implémentés en Java.

Jena offre une API pour la gestion des données RDF, RDFS, OWL du web sémantique et une structure relationnelle pour le stockage persistant des données RDF, RDFS et OWL. Jena fournit également un parseur RDF/XML, et des outils pour migrer des ontologies RDF dans d'autres formats.

Le modèle Jena est fait d'une hiérarchie d'interfaces et de classes qui implémentent ces dernières.
Nous avons synthétisé ce modèle sous forme de diagramme de classes UML comme l'indique la figure suivante (figure 9) :

[6] http://jena.sourceforge.net/

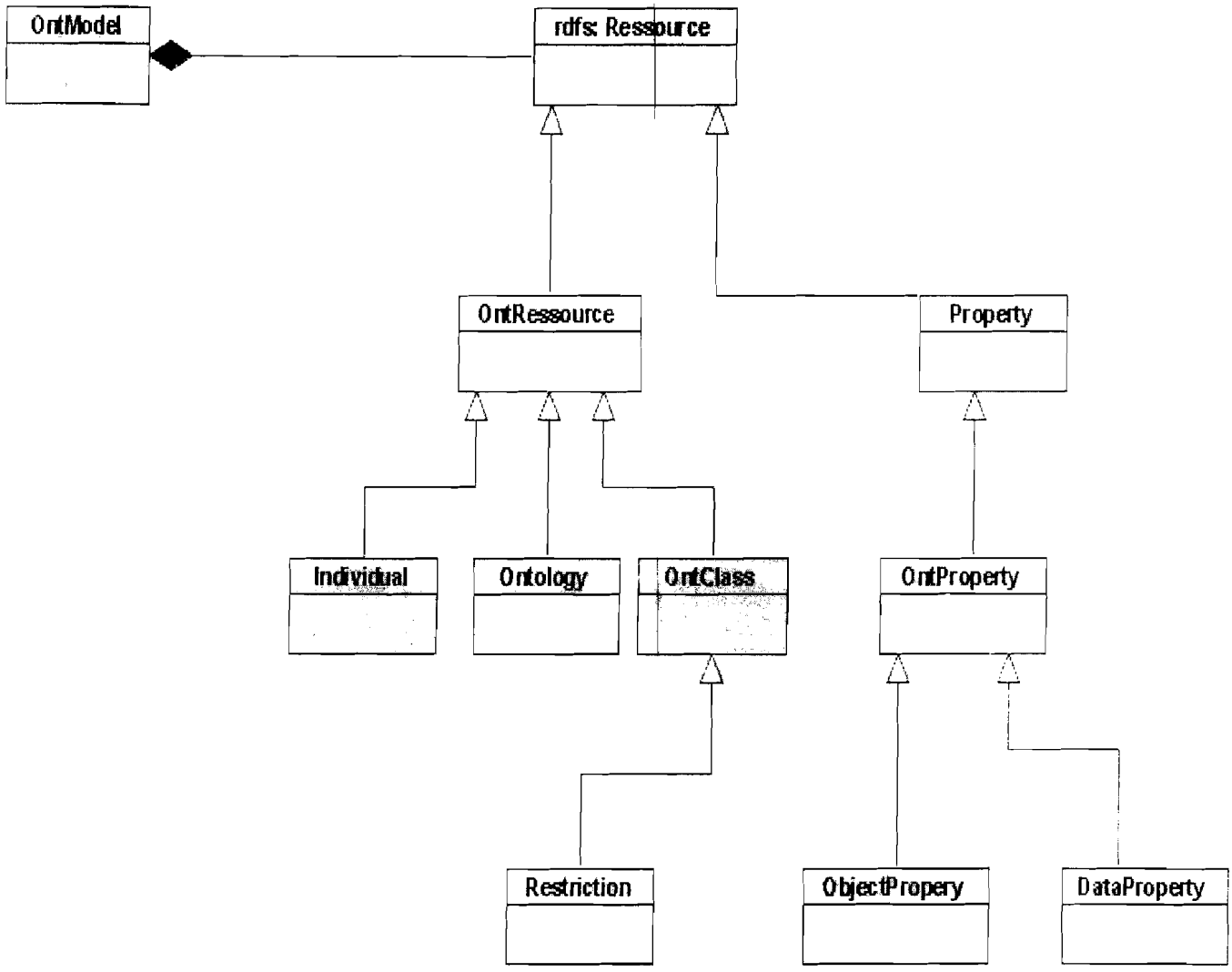

Figure 9 : Le modèle simplifié de Jena

Les concepts que nous aurons à extraire du modèle Jena vont de l'interface ***rdfs : Ressource*** *qui est liée à un modèle d'ontologie* ***ontModel*** *et qui est à la racine de tous les autres concepts clés (****Ontology, OntClass, OntProperty****).*

4.1.2. Modèle OWL API

OWL API[7] est également un framework open source et disponible sous licence LGPL. Son développement a surtout vu le jour à l'université de Manchester mais plusieurs autres contributions d'autres structures sont à mettre à son actif. Il est recommandé par le consortium W3C pour les travaux du web sémantique.

[7] http://owlapi.sourceforge.net/

Cette API JAVA est fortement liée au modèle d'ontologie OWL qui est à nos jours le standard en matière de représentation d'ontologies. Tout comme Jena, OWL API offre plusieurs méthodes pour manipuler (charger, lire, écrire, manager, parser,...) des ontologies.

L'architecture simplifiée de ce modèle donne la figure 10 ci-dessous :

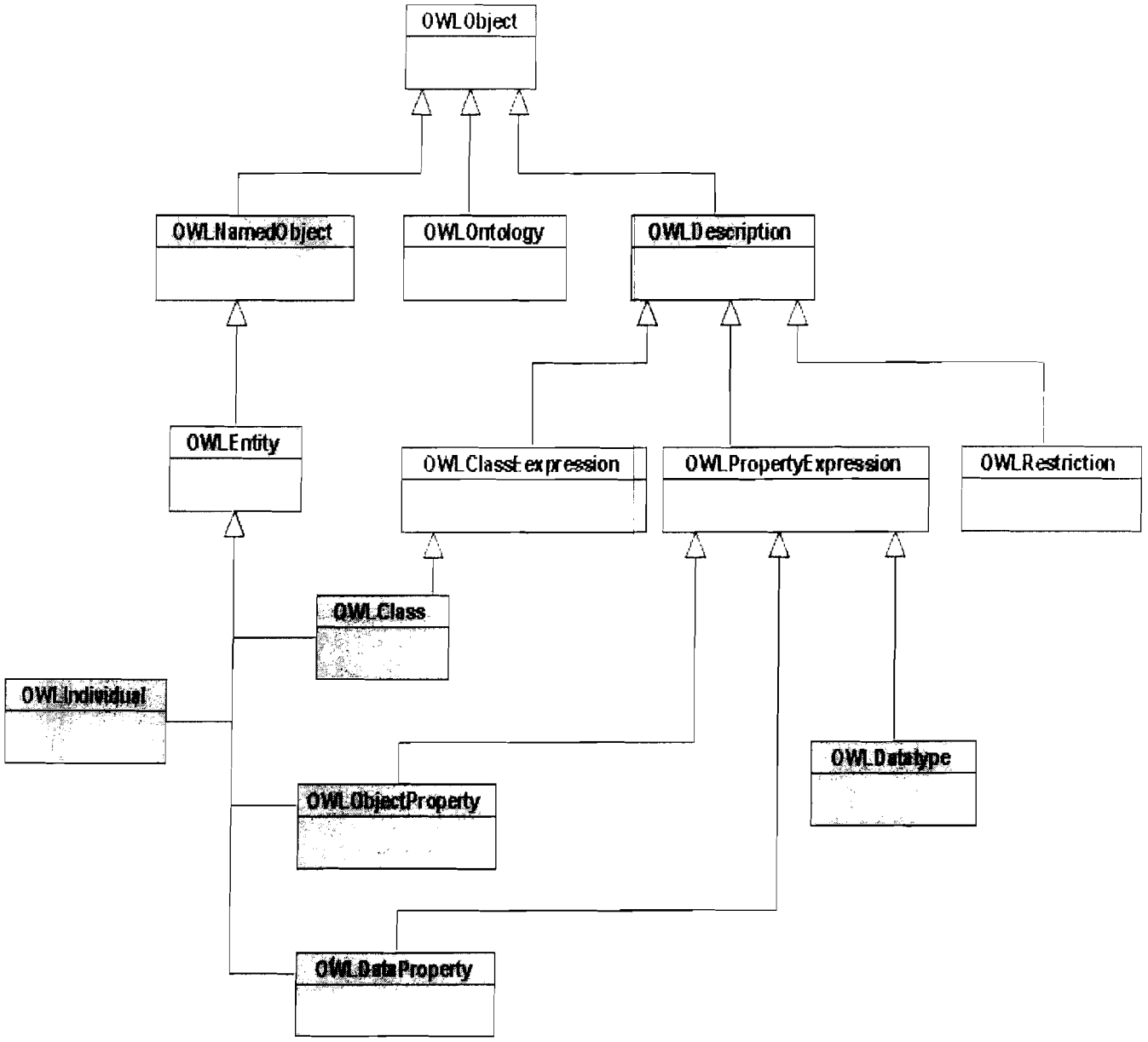

Figure 10 : Le modèle simplifié de OWL API

4.2. Modèle OWLConverter

La première étape de notre approche sera d'extraire des Framework Jena et OWL API les ressources et fonctionnalités dont nous aurons besoin pour la mise en place de notre API. De nos jours, ces Framework sont les plus utilisées et les plus complètes. Ce travail va consister à construire pour chaque ressource indispensable à notre API et contenue dans Jena ou OWL API, une nouvelle ressource qui nous sera propre.

Cette construction est faite en tenant compte de l'architecture du modèle OntoQL pour permettre une interaction plus fluide entre ces modèles et ainsi anticiper sur les ressources à extraire de Jena ou OWL API.

L'architecture de ces nouvelles ressources se présentera pour le cas d'une ***classe ontologie*** comme l'indique la figure 11 ci-dessous:

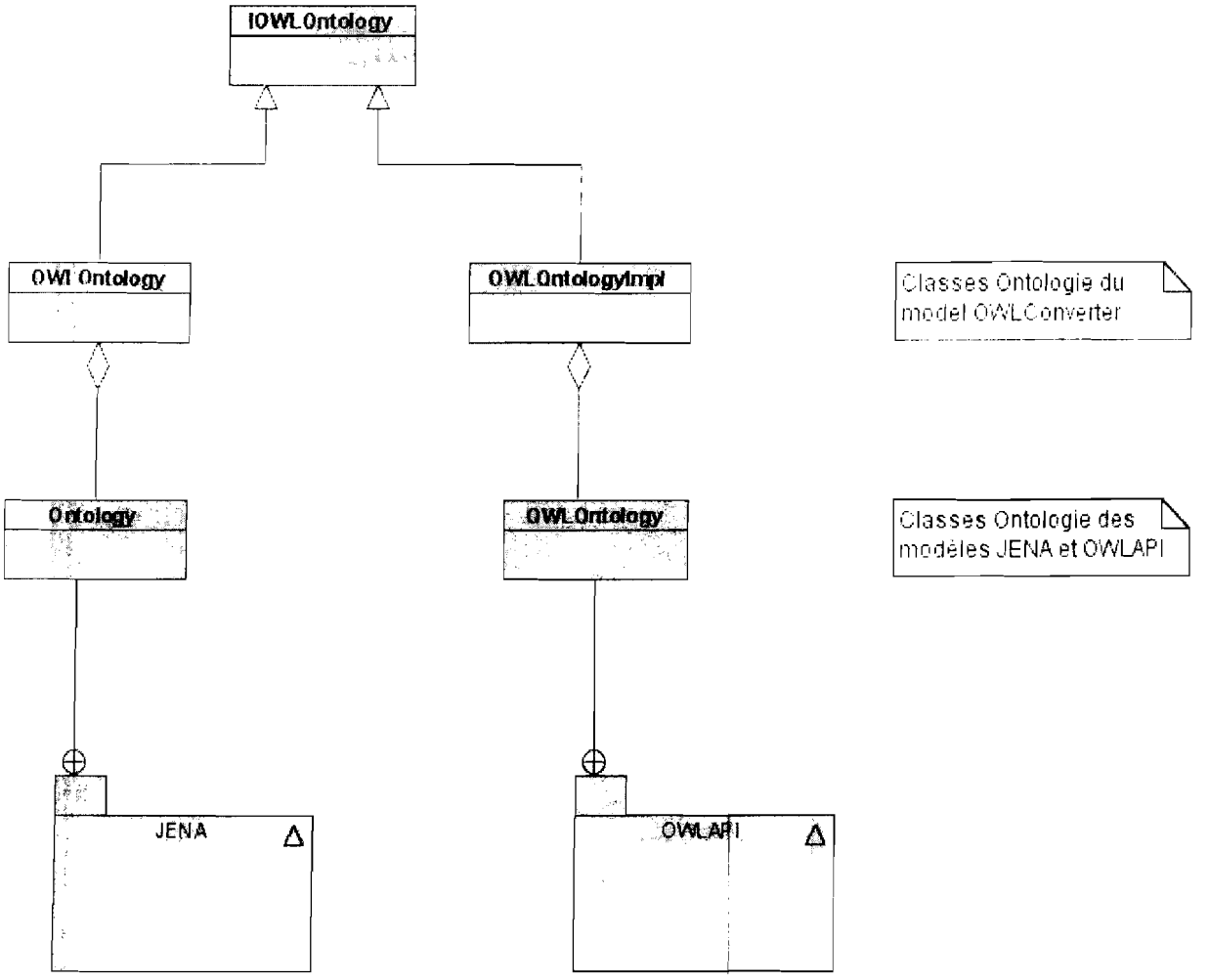

Figure 11 : Architecture des ressources de l'API : Cas d'une classe ontologie.

*Explication : Dans cette figure, le modèle OWLConverter aura deux classes « **ontologie** » : OWLOntology issue de l'adaptation à la classe Ontology de JENA et OWLOntologyImpl issue de la classe OWLOntology de OWL API. Pour rendre ces manœuvres transparentes à l'utilisateur de notre API, nous encapsulons ces deux classes dans une même interface nommée ici IOWLOntology.*

Pour tous les autres concepts ou ressources de notre API, nous construirons des correspondants comme dans la figure 11 ci-dessus. Ce qui va donner l'architecture globale dépeinte dans la figure 12 si on s'en tient aux principaux concepts de ***OWLConverter***.

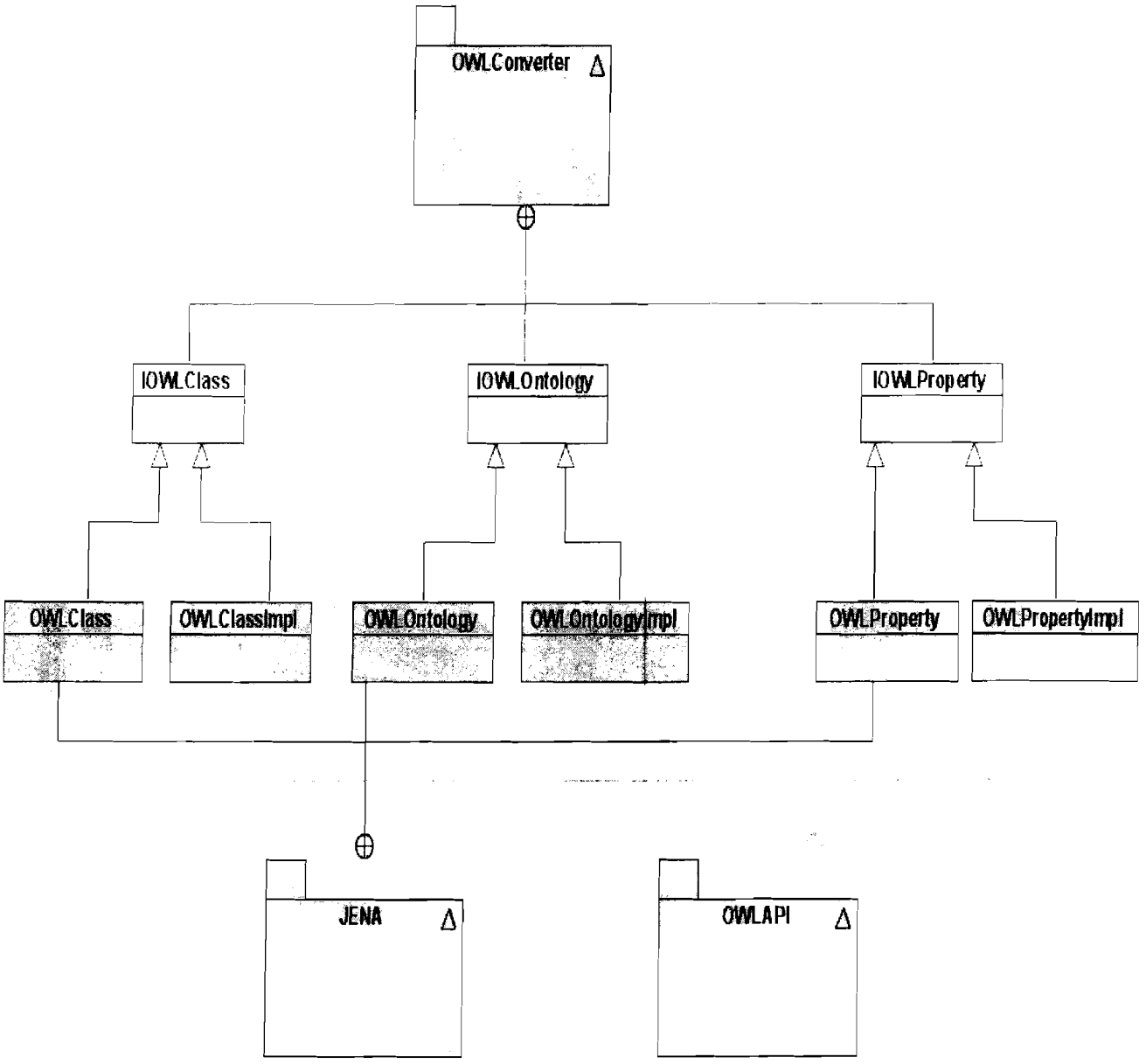

Figure 12 : L'architecture globale de l'API OWLConverter

4.3. Règles de transformation des ontologies OWL en PLIB

Dans cette partie, nous présentons les règles de transformation des concepts d'ontologies du modèle OWL en PLIB. Nous donnons les règles de conversion des classes, de l'héritage simple et multiple, ensuite nous traitons la conversion des propriétés, des métadescripteurs et en fin la transformation en PLIB des types de données.

4.3.1. Représentation d'une classe

Une classe du modèle OWL, notée OWLClass, est transformée en une classe PLIB (PLIB.item_class). La figure 13 illustre cette transformation.

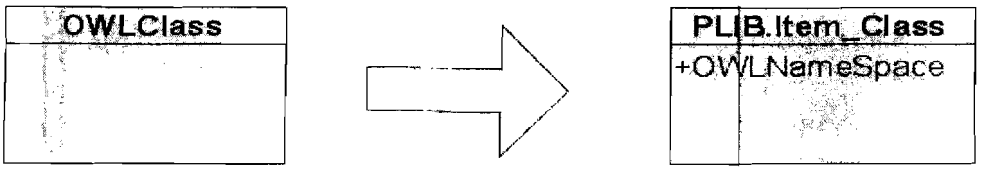

Figure 13 : Transformation d'une classe du modèle OWL en classe du modèle PLIB

Dans la description d'une classe du modèle OWL, il apparait obligatoirement un espace de nom ou ***namespace*** ce qui n'est pas le cas pour une classe en PLIB. Ainsi, une nouvelle propriété de description, appelée *OWLNamespace*, portant comme valeur l'espace de nom de la classe issue du modèle OWL, est ajoutée à la classe PLIB obtenue suite à la transformation pour ne pas perdre la sémantique de la classe de départ.

La transformation d'une classe du modèle OWL en classe PLIB modifie le modèle PLIB, car on ajoute un métadescripteur, ici *OWLNamespace*, à la classe PLIB pour garder le *namespace* de la classe du modèle OWL.

Il existe dans toute ontologie OWL une superclasse, nommée ***Thing***, dont toutes les autres classes sont des sous-classes. Ceci nous amène directement au concept d'héritage, disponible à l'aide de la propriété ***subClassOf***.

Nous présenterons, dans le paragraphe suivant, les règles de transformation de cette relation du modèle OWL en PLIB.

4.3.2. Représentation du principe d'héritage

La relation de l'héritage en OWL, comme en PLIB, permet à une classe donnée de reprendre toutes les propriétés d'une autre classe. La hiérarchie (l'héritage), dans le modèle OWL peut être simple ou multiple, en revanche, PLIB n'autorise que l'héritage simple.

Nous présentons la transformation de la relation de l'héritage dans le cas de la hiérarchie simple et multiple.

4.3.2.1. Cas d'une hiérarchie simple

Lorsqu'une classe OWL possède une seule superclasse (Pour les classes directement reliées à la super classe racine Thing, appelées en OWL ***classes root*** par exemple), la conversion de la relation de l'héritage en PLIB s'opère sans changement. La figure 14 illustre cette transformation.

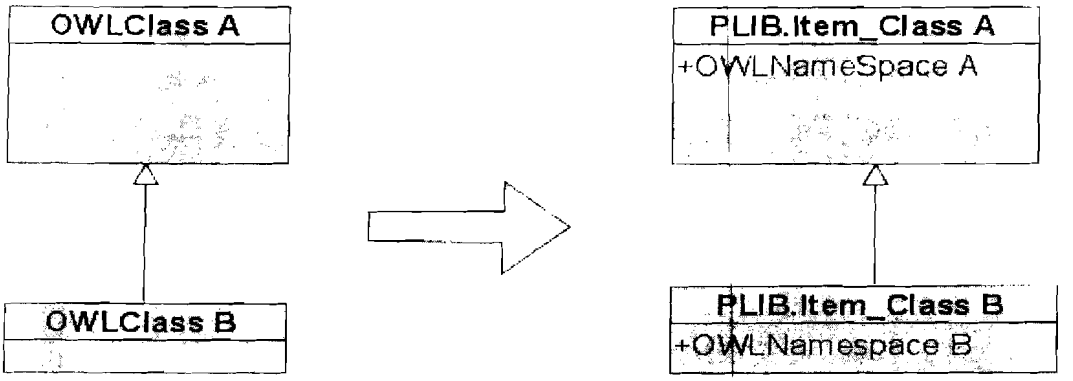

Figure 14 : Transformation de la relation d'héritage simple du modèle OWL en PLIB

4.3.2.2. Cas d'une hiérarchie multiple

Le modèle PLIB n'autorise pas l'héritage multiple. Toute fois, le mécanisme d'importation qui caractérise PLIB peut être exploité pour remplacer l'absence de hiérarchie multiple dans ce modèle d'ontologies. En effet, le mécanisme d'importation autorise une classe, en PLIB, à importer des propriétés d'autres classes à l'aide du mécanisme ***is_case_of***.

La relation d'héritage multiple en OWL est transformée ainsi : une seule superclasse, dans le modèle OWL est choisie pour être la superclasse directe dans le modèle PLIB, tandis que les autres superclasses vont être liées par la relation d'importation, avec la classe PLIB résultante. La figure 15 ci-dessous décrit la conversion de l'héritage multiple d'OWL en PLIB.

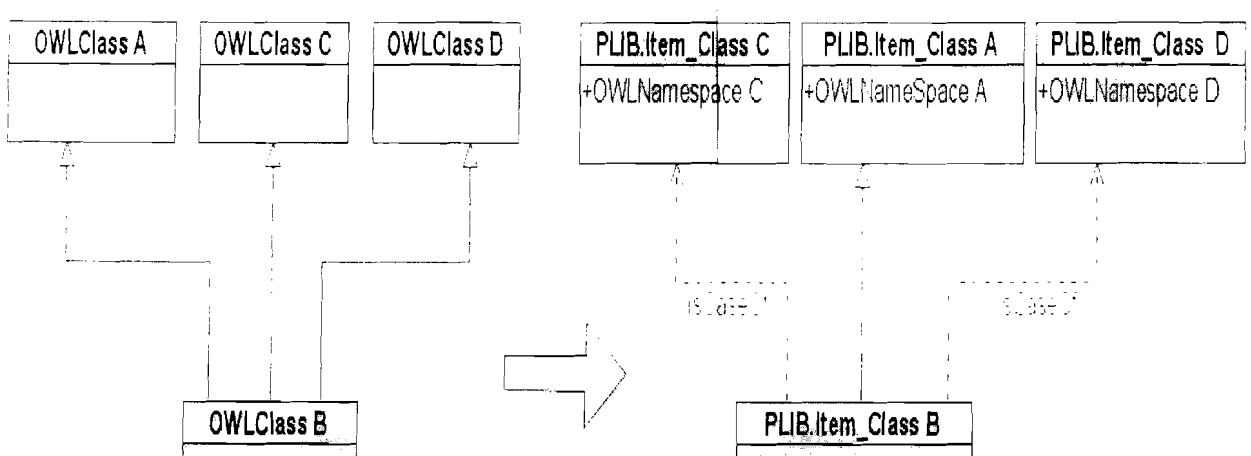

Figure 15 : Transformation de la relation d'héritage multiple du modèle OWL en modèle PLIB

Explication : Dans le modèle OWL, la classe OWLClass B hérite de trois classes qui sont OWLClass A, OWLClass C et OWLClass D. Lors de la conversion vers le modèle PLIB, la classe PLIB.item_class B, obtenue par transformation de la classe OWLClass B, hérite seulement de la classe PLIB.item_class A et importe les propriétés des classes PLIB.item_class C et D:PLIB.item_class D.

Maintenant que l'on sait écrire des classes OWL, il ne manque plus que la capacité à exprimer des faits au sujet de ces classes et de leurs instances. C'est ce que permettent de faire les propriétés OWL.

4.3.3. Représentation d'une propriété

De façon générale, une propriété OWL appelée OWLProperty est convertie en une propriété PLIB (*PLIB.non_dependant_P_DET*) comme représentée dans la figure 16 ci-dessous.

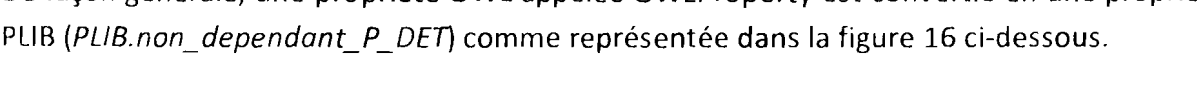

Figure 16 : Représentation de la conversion d'une propriété OWL en PLIB

Tout comme lors de la transformation d'une classe, une nouvelle propriété de description, OWLNamespace, portant comme valeur le namespace de la propriété issue du modèle OWL, est ajoutée à la propriété PLIB obtenue suite à la transformation. Ce nouveau métadescripteur permet de garder la sémantique de la propriété de départ.

OWL fait la distinction entre deux types de propriétés :

- les propriétés d'objet (**ObjectProperty**) permettent de relier des instances de classes (individus) à d'autres instances ;
- les propriétés de type de donnée (**DataTypeProperty**) permettent de relier des individus à des valeurs de données.

4.3.3.1. Cas d'une propriété d'objet (OWLObjectProperty)

Une propriété *OWLObjectProperty* dont le codomaine référence une instance d'une classe, est convertie en une propriété de type objet (référence) en PLIB. *OWLObjectProperty* est un sous type de *OWLProperty*. La figure 17 représente la transformation d'une propriété *OWLObjectProperty* en une propriété PLIB.

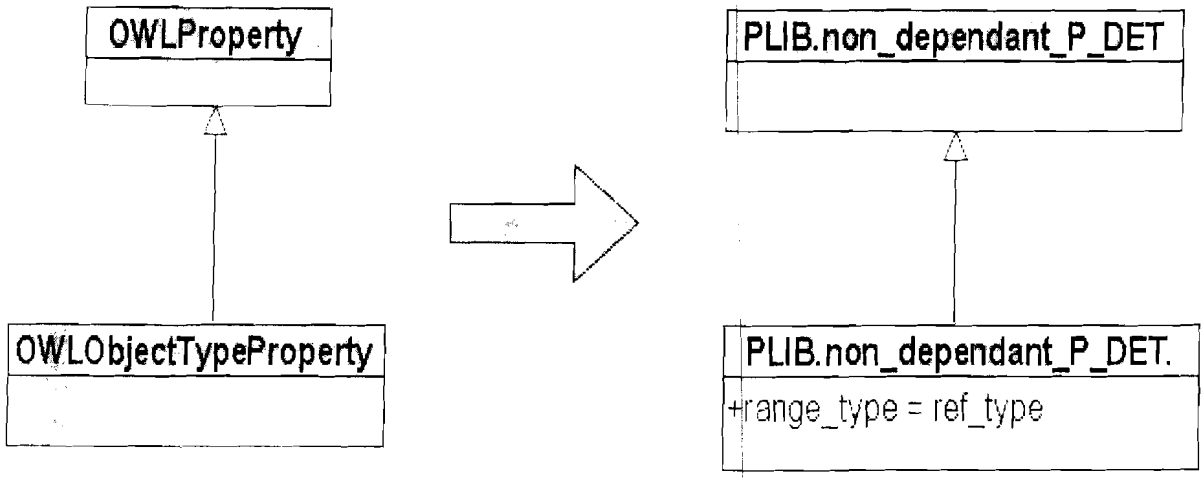

Figure 17 : Transformation d'un OWLObjectProperty en une propriété de type référence PLIB

Explication : L'attribut range_type désignant le type du codomaine de la propriété qui est dans ce cas de type référence, est déjà présent dans le modèle PLIB et ne constitue donc pas un nouvel attribut à ajouter au modèle lors de la transformation.

4.3.3.2. Cas d'une propriété de type donnée (OWLDataTypeProperty)

Une propriété *OWLDatatypeProperty* est convertie en une propriété de type simple dans le modèle PLIB. *OWLDatatypeProperty,* dans OWL, est un sous type de *OWLProperty* et représente les propriétés dont le codomaine (range) est de type simple (entier, chaine de caractères, date, booléen, ...). La figure suivante (figure 18) illustre la transformation d'une propriété *OWLDataTypeProperty*.

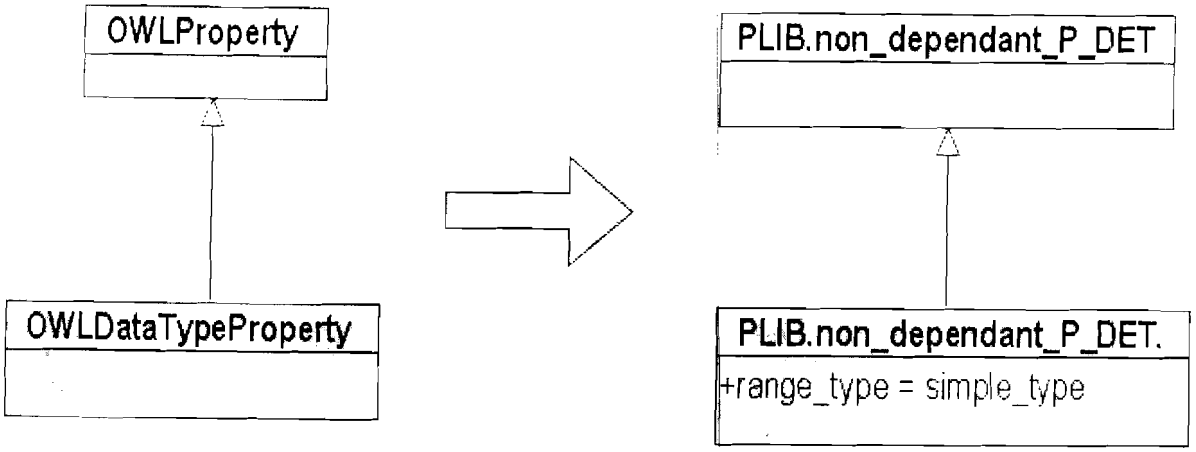

Figure 18 : Transformation d'un OWLDataTypeProperty en une propriété de type simple PLIB

Les modèles d'ontologies OWL et PLIB représentent les types de données de manières différentes. Pour la création concrète d'une propriété *OWLDataTypeProperty* dans le modèle PLIB, on doit déterminer de façon précise d'abord à quel type simple de données correspond cette *OWLDataTypeProperty*. Nous montrerons dans le paragraphe suivant, les transformations des types de données simples du modèle OWL en PLIB.

4.3.4. Conversion des types de données simples

Les types de données simples utilisés dans le modèle OWL sont ceux de XML Schema (xsd:int, xsd:string, xsd:date,...). Nous avons choisi de donner la conversion des types de données les plus utilisées que nous retrouvons dans le tableau 5 ci-dessous.

Tableau 5 : Correspondance des types de données simples du modèle OWL en PLIB

OWL	PLIB
xsd:Boolean	boolean_type
xsd:int	int_type
xsd:float / xsd:double	real_type
xsd:string	string_type

Les propriétés ainsi que les classes possèdent des attributs de description qu'on appelle des métadescripteurs. La transformation de ces derniers du modèle OWL en PLIB est expliquée dans le paragraphe suivant.

4.3.5. Conversion des Métadescripteurs

Les métadescripteurs permettent de décrire plus en détails les classes et les propriétés. Ces derniers, dans le modèle OWL, sont soit des ***RDFSLabels*** ou des ***RDFSComments***. Ces deux métadescripteurs correspondent, pour nous, respectivement aux descripteurs ***shortName*** et ***definition*** du modèle PLIB comme l'indique le tableau suivant :

Tableau 6 : Mise en correspondance entre métadescripteurs PLIB et OWL

OWL	PLIB
RDFSLabels	ShortName
RDFSComments	Definition

Après la phase de conception où nous avons abordé les règles de transformation du modèle OWL en PLIB, Nous verrons dans le point suivant la mise en œuvre de ces dernières en nous basant sur notre API construite à Partir de JENA et OWL API.

V.MISE EN ŒUVRE ET VALIDATION

5.1. Outils et technologies utilisés

Pour la réalisation des règles de transformation vues à la phase de conception, plusieurs outils et technologies existants au LISI ont été utilisés. Nous les présentons en deux groupes : les outils pour l'organisation et la programmation et les outils de tests sur les ontologies.

5.1.1. Programmation et organisation

Pour la programmation, nous avons utilisé le langage **Java** qui est le principal langage de développement utilisé par l'équipe IDD au sein de laquelle nous avons travaillé. Quant à l'environnement, c'est l'Interface de Développement Intégrée **Eclipse** qui est utilisée.

Aussi, au sein du LISI, un certain nombre d'outils d'intégration continue, permet d'assurer l'organisation et la qualité des applications en cours de développement. Ces logiciels sont exploités massivement dans les entreprises d'ingénierie logicielle :

- ***Apache Maven*** :

Maven est un outil de génération automatique des versions de logiciels Java. L'objectif recherché est de produire un logiciel à partir de ses sources, en garantissant le bon ordre de fabrication. Afin de décrire un logiciel, ses dépendances avec des modules externes et l'ordre à suivre pour sa production, Maven utilise un fichier appelé Project Object Model (POM). Maven peut ainsi télécharger directement depuis des « entrepôts » les dernières versions des logiciels dont il a besoin pour fabriquer le logiciel courant.

Maven impose une convention de nommage et d'arborescence des fichiers, permettant ainsi une grande cohérence dans l'organisation des projets développés. De plus, Maven possède un plugin Eclipse, permettant ainsi d'utiliser ces deux logiciels conjointement.

- ***Subversion :***

Subversion (SVN) est un système de gestion des versions. Son utilisation permet d'avoir sur le serveur de l'entreprise les dernières versions des fichiers sources des logiciels en cours de développement et permet le travail collaboratif, plusieurs personnes pouvant travailler sur un même fichier. Les développeurs travaillant sur le projet ont ainsi toujours à leur disposition les dernières sources du logiciel.

- ***Redmine :***

Redmine est une application Web de gestion de projets, permettant d'une part de suivre précisément l'évolution des versions : des demandes d'évolution peuvent être postées pour un projet donné, en attendant d'être résolues par l'un ou l'autre des développeurs du projet. D'autre part, *Redmine* propose un système de *Wiki* permettant de centraliser les connaissances sur tel ou tel outil.

L'application permet aussi une notification par mail des dernières évolutions. Elle peut gérer les feuilles de route, les calendriers et l'historique des versions. Elle permet aussi l'intégration avec des logiciels de suivi de versions, tels que *SVN*.

La figure 19 ci-dessous montre notre interface de travail *Redmine* où l'évolution de OWLConverter peut être suivie continuellement.

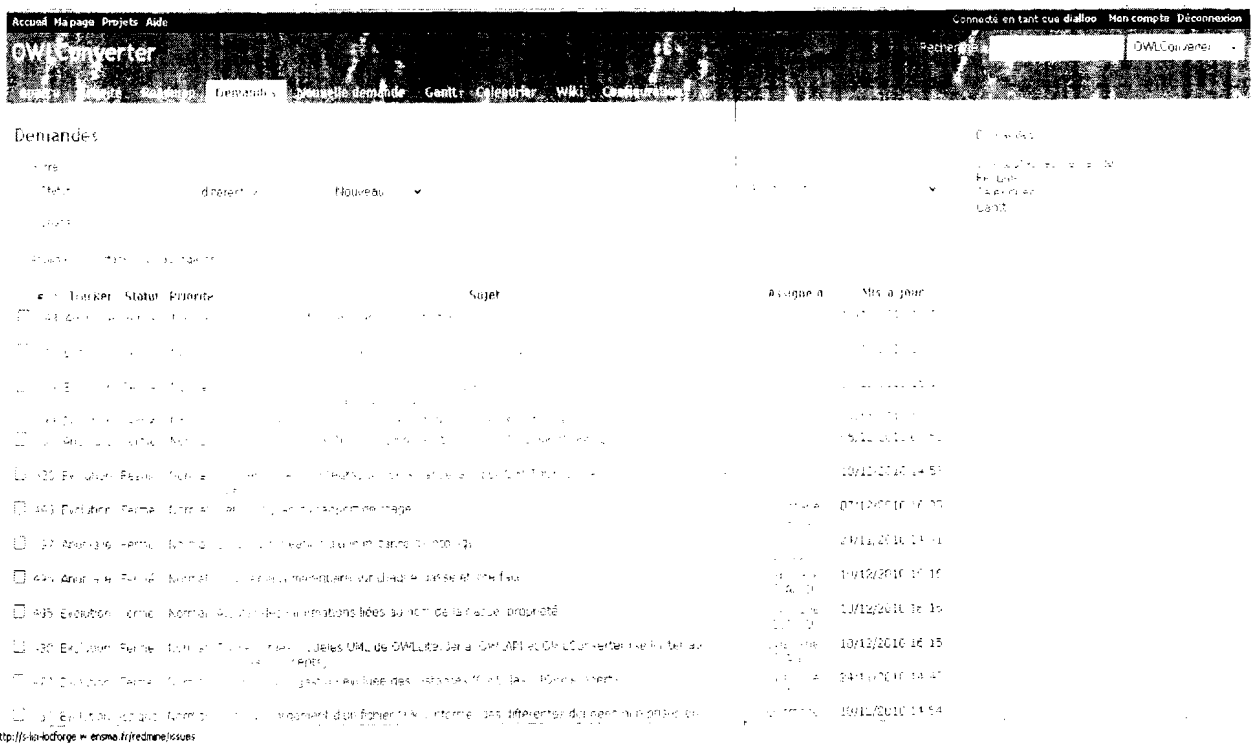

Figure 19 : la gestion de projets avec Redmine

5.1.2. Outils de tests sur les ontologies

- **Protégé**

Protégé [10] est une suite logicielle open source permettant de visualiser et/ou d'éditer les ontologies OWL, RDF(S). Il offre une interface assez conviviale pour manipuler facilement des ontologies. Nous l'avons utilisé tout au long du stage pour éditer nos ontologies de tests utilisées pour les tests unitaires de l'application.

La figure 20 présente l'interface de l'éditeur d'ontologie ***protégé.***

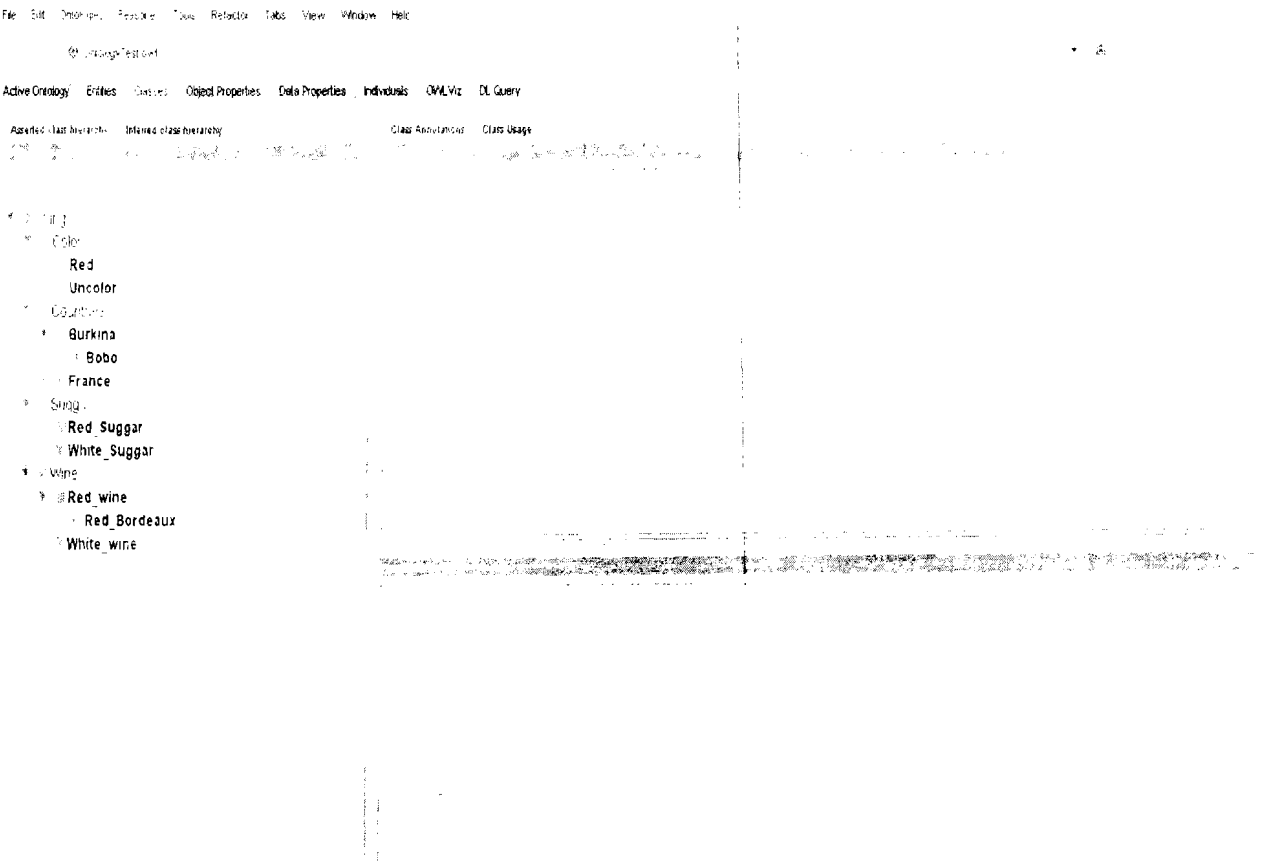

Figure 20 :L'édition d'ontologies avec protégé

- **OntoQL+**

OntoQL+ est un éditeur de requêtes dédié au langage OntoQL tout comme le *SQL+* l'est pour le langage SQL. Nous l'avons également utilisé tout au long du stage pour éditer des requêtes ou encore pour tester que les concepts et données OWL ont bien été insérés dans la BDBO OntoDB.

L'interface de l'éditeur *OntoQL+* se présente comme sur la figure 21 ci-dessous où les résultats de l'exécution des requêtes saisies dans la zone 1 s'affichent dans la zone 2.

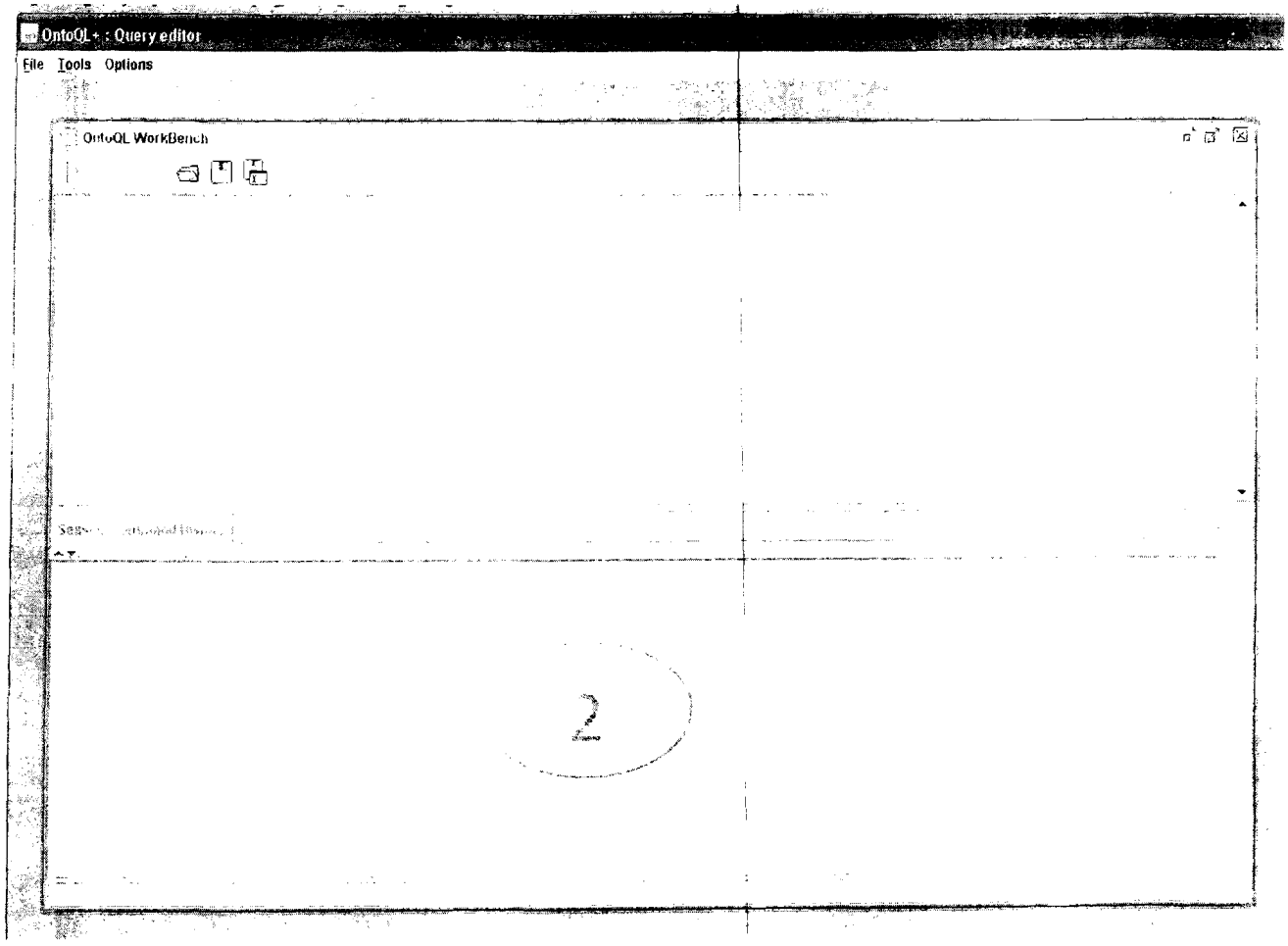

Figure 21 : Edition de requêtes OntoQL avec OntoQL+

5.2. Mise en œuvre de l'API OWLConverter

Pendant la phase de conception, les règles de transformation du modèle OWL en PLIB ont été définies. Les paragraphes de cette section expliquent, en détail, les procédures d'implémentation de toutes les règles de transformation des ontologies OWL en PLIB en utilisant notre API basée sur Jena et OWL API. Il s'agira ici d'utiliser cette API pour extraire d'une ontologie OWL tous les concepts et toutes les données qu'elle contient afin de les stockés dans la BDBO du modèle PLIB c'est-à-dire OntoDB.

5.2.1. Création de l'ontologie

La création d'une ontologie PLIB commence d'abord par la création de l'espace de nom (*namespace*) de cette ontologie dans OntoDB. Ainsi, nous extrayons de l'ontologie OWL, son *namespace* à partir de la méthode ***getNamespace*** de l'interface ***IOWLOntology*** de notre API.

Si nous supposons la variable *myOntology* de type *IOWLOntology*, l'instruction suivante permettra de récupérer le *namespace* de *myOntology* :

myOntology.getNamespace

Dès qu'on a cet espace de nom (« http://lisi.ensma.fr/ » par exemple), nous pouvons créer l'ontologie dans OntoDB avec la requête OntoQL suivante :

INSERT INTO #ONTOLOGY (#NAMESPACE) VALUES ("http://lisi.ensma.fr/")

Après le *namespace* de l'ontologie, on peut ainsi commencer à créer les classes et les propriétés. Nous donnons, dans le paragraphe suivant, la démarche suivie pour créer une classe PLIB à partir d'une classe OWL.

5.2.2. Les classes

Pour créer les classes PLIB et les stocker dans OntoDB, il faut récupérer, à partir d'OWL, toutes les informations nécessaires pour la création de ces classes, particulièrement le nom ou l'URI, les superclasses, les commentaires et les labels. Notre API permet de charger ces données avec les interfaces et les méthodes qui y ont été définies.

5.2.2.1. Création d'une classe

La création d'une classe PLIB sous OntoDB est précédée par la récupération de l'URI ou simplement du nom de la classe OWL avec la méthode ***getURI*** ou ***getLocalName*** de l'interface ***IOWLClass***. Ces méthodes permettent de récupérer, respectivement l'URI ou juste le nom d'une classe OWL qui va servir à formuler la requête OntoQL pour créer cette même classe sous OntoDB.

Si nous supposons la variable ***myClass*** de type *IOWLClass* et que ***myClass.getLocalName*** renvoie « Thing », la requête OntoQl de creation de la classe Thing en PLIB sera :

CREATE #CLASS Thing

5.2.2.2. La relation d'héritage

Comme dit précédemment dans la phase de conception, dans le modèle OWL, il existe pour chaque ontologie, une superclasse racine par rapport à laquelle toutes les autres classes sont des sous classes : c'est la classe ***Thing***. Nous avons exploité ce fait au cours de l'implémentation, ce qui résout au passage le problème de transformation de l'héritage multiple dont on parlait dans la phase de conception. Cette classe est obtenue par la méthode ***getThing*** de l'interface *IOWLOntology*.

Ainsi à partir du Thing, on peut avoir les sous classes directes de chaque classe en utilisant la méthode ***getSubClasses*** de l'interface *IOWLClass*. Et à partir de cette information, on crée chaque classe en OntoQL avec la requête suivante :

CREATE #CLASS className UNDER superClass

superClass est le nom de la superclasse à partir de laquelle on a appelé la méthode *getSubclasses* et ***className***, le nom de la classe (dans la liste) qu'on est entrain de créer.

Exemple: ***Create #Class Wine under Thing*** (ici la classe Wine est sous classe donc hérite de la classe superclasse Thing).

5.2.2.3. Les métadescripteurs

Chaque classe est décrite par un ensemble de propriétés de description (métadescripteurs) qui lui donnent une signification.

Les métadescripteurs qui existent en OWL sont les commentaires (*RDFSComment*) et les labels (*RDFSLabel*). Les méthodes ***getComment*** et ***getLabel*** de l'interface *IOWClass* permettent respectivement d'extraire les commentaires et les labels d'une classe définie dans le modèle OWL.

Les labels sont transformés en *shortName* et les commentaires sont convertis en *definition*. On associe les métadescipteurs à la classe lors de la création de cette dernière, en donnant à chaque métadescripteur en PLIB la valeur homologue qui lui correspond en OWL. Nous donnons un exemple de requête OntoQL qui montre l'association des métadescripteurs à une classe.

```
CREATE #Class Wine
        UNDER Thing
                (DESCRIPTOR (#definition[fr] = 'Vin'))
```

*On associe à la classe Wine un métadescripteur « **definition** » en langue française **[fr]** pour spécifier qu'on parle du « **vin** ».*

5.2.3. Les propriétés

Pour créer des propriétés PLIB à partir de OWL, on procédera de la même manière que lors de l'implémentation de la transformation des classes. C'est-à-dire nous utilisons des méthodes de l'API pour extraire les informations utiles à cette création du modèle OWL et nous créons les propriétés dans OntoDB.

Il faut noter que dans le modèle PLIB, la définition des propriétés se fait simultanément avec celle des classes.

5.2.3.1. Création d'une propriété

La méthode ***getPropLocalName*** ou ***getPropURI***, de l'interface ***IOWLProperty***, permet de récupérer respectivement le nom ou l'URI de la propriété dans le modèle OWL. Cet URI ou ce nom permet de créer la propriété équivalente en PLIB.

Voici un exemple de requête de création d'une classe incluant la création d'une propriété :

CREATE #Class Wine UNDER Thing (

PROPERTIES (isRed REF (Color), alcoholPercentage INT))

REF signifie que la propriété référence une instance d'une classe (ObjectProperty). Ainsi la classe Wine possède deux propriétés, isRed référençant une instance de la classe Color et alcoholPercentage qui est un entier (INT).

5.2.3.2. Définition du domaine et du codomaine

Le domaine de la propriété est la classe de l'ontologie dans laquelle une propriété est définie. La méthode ***getDomain*** de l'interface *IOWLProperty* permet de retourner le domaine d'une propriété.

Le codomaine (range) d'une propriété représente son type, ce dernier peut être simple (entier, date, ...) ou objet (instance d'une classe). La méthode ***getRange*** de l'interface *IOWLProperty* retourne le codomaine d'une propriété, qu'elle soit de type simple ou objet.

Reconsidérons l'exemple précédant:

CREATE #Class Wine UNDER Thing (

PROPERTIES (isRed REF (Color), alcoholPercentage INT))

La classe Wine représentera le domaine des propriétés isRed et alcoholPercentage. Tandis que la propriété isRed aura pour codomaine Color et celui de alcoholPercentage sera INT (Integer).

5.2.3.3. Les métadescripteurs

Les propriétés, comme les classes, sont décrites, en OWL, par des commentaires et des labels. Les méthodes ***getComment*** et ***getLabel*** de l'interface ***IOWLProperty*** permettent respectivement de retourner les commentaires et les labels d'une propriété.

Une fois ces métadescripteurs extraits du model OWL, on peut les transformer en leurs équivalents en PLIB. L'exemple suivant montre comment attribuer les métadescripteurs à une propriété :

CREATE #Class RedWine UNDER Wine

DESCRIPTOR (#definition [fr] = 'VinRouge')

PROPERTIES (hasTop REF (BordeauxWine) ARRAY

DESCRIPTOR (#definition [en] = 'hasTop'))

*Explication : La propriété **hasTop** est décrite par le métadescripteur **definition** en langue anglaise **([en])**. On aurait pu le définir en langue française **[fr]** comme c'est le cas du métadescripteur de la classe **RedWine**.*

5.3. Validation

Tout au long du développement de OWLConverter, un accent particulier a été mis sur les tests de validation des différents modules afin de fournir au bout, une application respectant la politique qualité standard du développement Java et celle spécifique au laboratoire. En effet cette stratégie nous a permis d'identifier et de corriger les différentes anomalies laissées au fur et à mesure que l'on avançait dans le développement tout en assurant la non-régression des fonctionnalités de l'application. Ces tests sont, pour ce qui nous concerne, essentiellement les tests unitaires ; les tests d'intégrations n'ayant pas été explicitement mis en œuvre à l'étape actuelle du développement.

En effet, les tests unitaires sont devenus partie prenante d'un courant de pensée dont l'objectif est d'améliorer la qualité du logiciel. On peut ici décliner cette notion de qualité sur deux aspects :

• ***Le respect des exigences***. La première volonté d'amélioration de la qualité est de respecter les exigences. Nous entendons ici par exigence le contrat que doit implémenter une méthode de l'API. C'est-à-dire qu'au-delà du respect des normes de présentation du code (conventions de nommage, commentaires, indentation,..), ce critère veille à ce que chaque méthode (unité) de l'application fasse bien ce pourquoi elle a été implémentée.

• ***La réutilisabilité***. Cette notion est historiquement le principal souci du développement. L'idée sous jacente consiste à dire qu'il est préférable de réutiliser un composant testé et validé, dont nous sommes sûrs de la qualité plutôt que de recréer la même fonctionnalité avec le risque d'introduire d'autre bugs.

Nous avons utilisé la bibliothèque de tests unitaires, ***JUnit***, fourni par le langage Java pour vérifier, pendant la phase de tests, que les éléments fournis en entrée pour chaque méthode donnent effectivement le résultat attendu. Ainsi nous avons testé de bout en bout depuis le passage d'un fichier OWL en paramètre, que l'extraction de concepts et données, leur transformation en PLIB ainsi que leur insertion dans OntoDB à travers des requêtes OntoQL se déroulaient normalement.

BILAN ET PERSPECTIVES

L'objectif principal du stage était d'étudier et d'implémenter les règles de transformations des ontologies représentées en OWL vers PLIB en utilisant une API construite en amont sur la base des modèles JENA et OWL API.

Pour ce faire, nous étions amenés à étudier et comprendre la notion d'ontologie, ses caractéristiques et ses domaines d'utilisation. Ensuite, il a fallu comprendre les spécifications des deux modèles d'ontologies OWL et PLIB afin d'exhiber les différences de représentation des concepts d'ontologies et des données de ces deux modèles. Allant de cette base, nous avons examiné les APIs JENA et OWL API afin d'en extraire que les concepts et ressources dont nous aurons besoin dans la mise en œuvre des règles de transformation, et les encapsuler dans notre propre API.

Nous avons aussi étudié l'architecture de la BDBO OntoDB, développée au LISI ainsi que le langage d'exploitation de ce BDBO, à savoir le langage de requêtes OntoQL.

Lors de la mise en œuvre, nous avons utilisé notre API pour charger et gérer des ontologies. Les méthodes fournies par cette API ainsi que l'utilisation du langage OntoQL nous ont grandement servi pour l'implémentation des règles de transformation des concepts d'ontologies du modèle OWL au modèle PLIB. En particulier, nous avons développé un programme qui permet de transformer les concepts et données d'ontologies du modèle OWL au modèle PLIB.

Le travail réalisé ouvre des perspectives allant dans le sens d'améliorer d'abord *OWLConverter* à travers l'extension des concepts de OWL Lite à OWL DL et implémenter l'inverse des règles de transformations c'est-à-dire du modèle PLIB vers OWL en suivant la même logique. Aussi, au sein du LISI, il existe un nouveau format de données appelé ***OntoML*** et correspondant à la norme ISO13584-32. Il serait intéressant d'étendre *OWLConverter* de sorte à y intégrer ce nouveau format basé sur XML et qui est utilisé comme format d'échange de données PLIB. Enfin, le LISI travaille sur une suite logicielle pour la visualisation et l'édition d'ontologies basées sur le modèle PLIB appelée ***OntoWebStudio***. Une autre perspective serait d'étudiée la possibilité de joindre *OWLConverter* à *OntoWebStudio* de sorte à lui associer des IHM (Interfaces Homme Machine) pour rendre plus intuitives les conversions d'un modèle d'ontologie à un autre.

Sur le plan personnel, les quatre mois de stage effectués au sein du LISI nous ont été d'un apport certain tant sur le plan organisationnel que sur le plan technique. En effet, sur le plan organisationnel, nous avons fais la découverte de la gestion d'un projet informatique en équipe, avec un contrôle de qualité des sources et l'utilisation des outils d'intégration continue et de gestion de projets.

Sur le plan technique, ce stage nous a permis d'exploiter les connaissances acquises durant notre formation d'ingénieur d'une part et d'autre part, nous avons pu nous initier au domaine de la recherche scientifique dans un laboratoire de recherche en informatique qui rassemble d'éminents chercheurs du domaine de l'ingénierie de données. Notre séjour au LISI nous a permis de découvrir le domaine des ontologies et les technologies récentes liées à cette discipline (OWL, PLIB, OntoDB, OntoQL).

BIBLIOGRAPHIE

[1] Gruber T. R. (1993). A translation approach to portable ontology specifications.Knowledge Acquisition, 5(2):199–220.

[2] Jean S., Pierra G. and Aït-Ameur Y. (2007d). Domain Ontologies: A Database-Oriented Analysis. In Web Information Systems and Technologies, International Conferences, WEBIST 2005 and WEBIST 2006. Revised Selected Papers, Lecture Notes in Business Information Processing, pages 238–254. Springer Berlin Heidelberg.

[3] Haav H.-M. and Lubi T.- L. (2001). A Survey of Concept-based Information Retrieval Tools on the Web. In Proceedings of the 5th East European Conference on Advances in Databases and Information Systems (ADBIS'01), pages 29–41.

[4] Tetlow P., Pan J., Oberle D., Wallace E., Uschold M., and Kendall E. (2005). Ontology Driven Architectures and Potential Uses of the Semantic Web in Systems and Software Engineering. World Wide Web Consortium. http://www.w3.org/2001/sw/ BestPractices/SE/ODA/.

[5] Dehainsala H., Pierra G., Bellatreche L., and Aït-Ameur, Y. (2007). Conception de bases de données à partir d'ontologies de domaine : Application aux bases de données du domaine technique. In Actes des 1ère Journées Francophones sur les Ontologies (JFO'07), pages 215–230.

[6] Goh C. H. (1997). Representing and reasoning about semantic conflicts in heterogeneous information systems. PhD thesis, MIT Sloan School of Management.

[7] Manola F. and Miller E. (2004). RDF Primer. World Wide Web Consortium (W3C). http://www.w3.org/TR/rdf-primer.

[8] Lacot X. (2005) Introduction à OWL, un langage XML d'ontologies Web, pages 14 – 27.

[9] Jean S. (2007) OntoQL, un langage d'exploitation des bases de données à base ontologique, thèse, Université de Poitiers, pages 102 – 137.

[10] Université de Stanford. The Protégé Ontology Editor and Knowledge Acquisition System. http://protege.stanford.edu/ (en ligne au 30 Décembre 2010).

Youness BAZHAR, Une approche d'alignement de modèles d'ontologies basée sur la transformation de modèles : Application aux modèles PLIB et OWL, rapport de stage de fin d'étude, 79, LISI, 2009

Alain BERNARD OntoWebStudio, une suite logicielle pour la visualisation et l'édition des ontologies basées sur le modèle PLIB, rapport de stage de master, 71, LISI, 2010