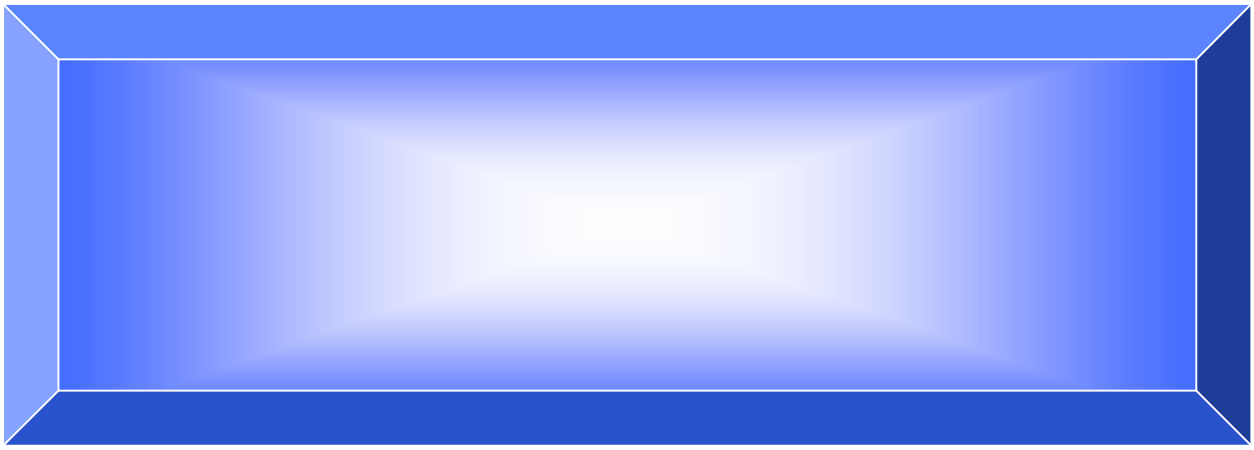
Master Sciences et Technologies
mention STIC (Sciences et Technologies de l'Information et de la Communication)



LISI ENSMA

Master Sciences et Technologies
mention STIC (Sciences et Technologies de l'Information et de la Communication)

VERS UNE INTEGRATION DES DIFFERENTES APPROCHES DE MODELISATION A BASE ONTOLOGIQUE : application aux modèles PLIB et OWL.

Présenté et soutenu par
FANKAM NGUEMKAM Chimène Jeannette

Mémoire de Stage recherche M2, spécialité F3I

effectué au
au **L**aboratoire d'**I**nformatique **S**cientifique et **I**ndustrielle **(LISI)**
de l'**E**cole **N**ationale **S**upérieure de **M**écanique et **d'A**érotechnique
(ENSMA) de Poitiers.

sous la direction de :
Pr. Yamine AIT-AMEUR
Pr. Guy PIERRA

SOMMAIRE

SOMMAIRE ..... **II**

REMERCIEMENTS ..... **IV**

RESUME ..... **V**

ABSTRACT ..... **VI**

INTRODUCTION GENERALE **2**

PARTIE I : PRESENTATION GENERALE **4**

INTRODUCTION **5**
CHAPITRE I : LES ONTOLOGIES EN INGENIERIE DES DONNES. **6**
I.1 LA NOTION D'ONTOLOGIE 6
I.2 QUELQUES DOMAINES D'APPLICATION DES ONTOLOGIES 7
I.3 POURQUOI INTÉGRER LES APPROCHES DE MODÉLISATION À BASE D'ONTOLOGIE ? 10
CHAPITRE II : LE MODELE D'ONTOLOGIE PLIB **12**
II.1 LES CARACTÉRISTIQUES D'UNE ONTOLOGIE PLIB 12
II.2 LES CONSTRUCTEURS DU MODÈLE PLIB [PIE 02] 12
II.3 GESTION DU CONTEXTE DANS LE MODÈLE PLIB 15
II.4 MODÈLE FORMEL D'UNE ONTOLOGIE PLIB [PIE 05] 16
II.5 ONTODB : UN MODÈLE D'ARCHITECTURE POUR LES BASES DE DONNÉES À BASE ONTOLOGIQUES 17
CHAPITRE III : LE LANGAGE D'ONTOLOGIE OWL **19**
III.1 RDF 19
III.2 RDFS 22
III.3 OWL 25
CHAPITRE IV : CONVERSION DE SCHEMA ENTRE MODELES DE DONNEES **40**
IV.1 CHOIX DU FORMALISME DE MODÉLISATION 41
IV.2 LE LANGAGE EXPRESS 42
IV.3 LE SCHÉMA EXPRESS-G D'OWL 44
CONCLUSION **50**

PARTIE II : ANALYSE ET CONCEPTION DES REGLES DE MAPPING DES MODELES PLIB ET OWL. **51**

INTRODUCTION **52**
CHAPITRE V : CONVERSION DE OWL VERS PLIB **53**
V.1 ANALYSE DE LA DÉMARCHE DE TRANSFORMATION 53
V.2 SPÉCIFICATION DU MAPPING D'UNE ONTOLOGIE OWL VERS LE MODÈLE PLIB 70
V.3 MISE EN ŒUVRE DE LA CORRESPONDANCE D'UNE ONTOLOGIE OWL VERS PLIB 72
CHAPITRE VI : CONVERSION DE PLIB VERS OWL. **74**
VI.1 CRITÈRE DE SÉLECTION DES ENTITÉS À CONVERTIR 74
VI.2 ANALYSE DE LA DÉMARCHE DE TRANSFORMATION 74
VI.3 EXTENSIONS ENVISAGÉES D'OWL 83
VI.4 SPÉCIFICATION DU MAPPING D'UNE ONTOLOGIE PLIB VERS OWL 84
VI.5 MISE EN ŒUVRE DE LA CORRESPONDANCE D'UNE ONTOLOGIE PLIB VERS OWL 86
CONCLUSION **89**

CONCLUSION GENERALE **91**

ANNEXES : SCHEMA EXPRESS DU MODELE PLIB **92**

ANNEXE 1 : LE MODÈLE EXPRESS-G DE PLIB : DICTIONNAIRE **93**
ANNEXE 2 : LE MODÈLE EXPRESS-G DE PLIB : BSU **94**
ANNEXE 3 : LE MODÈLE EXPRESS-G DE PLIB : CLASSE **95**
ANNEXE 4 : LE MODÈLE EXPRESS-G DE PLIB : PROPRIÉTÉ **96**
ANNEXE 5 : LE MODÈLE EXPRESS-G DE PLIB : TYPES DE DONNÉES **97**
ANNEXE 6 : LE MODÈLE EXPRESS-G DE PLIB : CONTRAINTES **98**

BIBLIOGRAPHIE **99**

TABLE DES MATIERES **101**

LISTE DES FIGURES **106**

LISTE DES TABLEAUX **107**

GLOSSAIRE **108**

REMERCIEMENTS

Je remercie tout d'abord le Seigneur tout puissant pour la grâce qu'il m'a accordé de mener à bien ce travail.

Mes remerciements les plus sincères vont :

- Au Pr. Guy PIERRA pour m'avoir donné l'opportunité de mener ces travaux au sein de son laboratoire. Je le remercie pour toute la confiance qu'il m'a témoignée, ses conseils, ses remarques et pour toutes les ressources qu'il a mises à ma disposition.

- Au Pr. Yamine AIT-AMEUR, je le remercie pour son encadrement, sa bonne humeur, ses remarques et les conseils qu'il m'a prodigués afin que je puisse mener à bien ce travail.

- A tous les membres du laboratoire : Hondjack, Stéphane JEAN, Dung, Loé, et, plus particulièrement aux membres du DEA F3I Mounira, Walid, Sybille, Nabil et Majid pour les bons moments passés en leur compagnie.

- A mes parents et à mes frères et pour leur soutien moral et leurs encouragements.

- A mes amis : Patrick, Franck Gérard et Emmanuel. Je les remercie pour leur soutien sans faille, leurs encouragements et leurs conseils.

RESUME

Les ontologies, définies par Gruber [Gruber 93] comme « une spécification explicite d'une conceptualisation », se sont imposées comme un moyen prometteur pour résoudre les problèmes de compréhension des données, et permettre aux programmes de reconstituer des raisonnements et des actions intelligentes sur les données dans les domaines aussi variés que l'intégration des données et la recherche sur le WEB, tout en restant le plus général possible.

Les ontologies sont une solution à bien des problèmes surtout dans le cadre de l'interopérabilité structurelle et sémantique entre des données issues de différentes sources. Cependant, il apparaît que pour un meilleur échange de l'information, toutes les personnes et les organisations devraient utiliser le même modèle d'ontologies. Cette situation est loin d'être atteinte, ni très réaliste. De ce fait, afin d'échanger les informations, il faudrait intégrer les différentes ontologies.

Ce stage porte sur l'intégration des modèles d'ontologies PLIB et OWL. Son objectif est d'identifier leurs différentes constructions puis, d'élaborer des règles permettant leurs intégrations.

***Mots clés* : ontologie, modélisation à base ontologique, intégration, PLIB, OWL.**

ABSTRACT

Ontologies defined by Gruber [Gruber 93] as "an explicit specification of a conceptualization", have become the more relevant way to solve data understanding problems and to allow programs to observe meaningful operations on data in various domains as data integration as well as search on WEB.

Ontologies are the answer to many problems like structural and semantic interoperability between data from different sources. However, it appears that for a best data exchange, all people and organizations should use the same ontology model. This situation is far from being reached and is utopist.

The present work is concerned by the integration of PLIB and OWL models. Its aim is to identify their constructs and to build rules enabling their mapping.

Keys words*: *ontology, ontology based modeling, integration, PLIB, OWL.

INTRODUCTION GENERALE

INTRODUCTION GENERALE

Parallèlement à l'explosion de l'information dans plusieurs domaines qui a marqué ces dernières années, des progrès considérables ont été observés dans le développement de méthodes de modélisation génériques des données et dans les technologies de recherche de l'information afin de manipuler la diversité des données et leur ampleur.

Les ontologies, définies par Gruber [Gruber 93] comme « une spécification explicite d'une conceptualisation », se sont imposées comme un moyen prométeur pour résoudre les problèmes de compréhension des données, et permettre aux programmes de reconstituer des raisonnements et des actions intelligentes sur les données dans les domaines aussi variés que l'intégration des données et la recherche sur le WEB, tout en restant le plus général possible.

Les ontologies sont une solution à bien des problèmes surtout dans le cadre de l'interopérabilité structurelle et sémantique entre des données issues de différentes sources. Cependant, la diversité des modèles d'ontologies existants pourraient re-ouvrir des problèmes d'échanges de données. Il apparaît que pour un meilleur échange de l'information, toutes les personnes et les organisations devraient utiliser le même modèle d'ontologies. Cette situation est loin d'être atteinte, ni très réaliste car pour diverses raisons (sociales, stratégiques, …), des personnes et des organisations différentes tendent généralement à utiliser différents modèles d'ontologies. De ce fait, afin d'échanger les informations, il faudrait soit intégrer les différentes ontologies, soit adopter un modèle d'ontologies unique.

Des efforts importants ont donc été réalisés pour concevoir des ontologies standards pour représenter et échanger l'information. Il s'est toutefois avérer que l'adoption d'une ontologie partagée par tous est non seulement difficile mais loin d'être atteinte. En effet, les ontologies sont généralement construites en fonction d'une vision particulière du monde. Ainsi, pour des personnes et des organisations utilisant des ontologies et ayant besoin d'échanger des informations, il apparaît un besoin de réconcilier leurs différentes ontologies.

Dans le cas d'une entreprise industrielle par exemple, il est facile d'imaginer le scénario suivant :

- Les employés ont besoin de réconcilier leurs ontologies personnelles avec celle de leur département ou même celle de toute l'entreprise.
- L'entreprise a besoin de réconcilier son ontologie avec celles de ses partenaires ou avec celles d'autres entreprises exerçant dans le même domaine.

Plus précisément, un domaine donné peut avoir plusieurs ontologies décrites en utilisant des langages différents et, basées sur différentes logiques. De ce fait, l'intégration des données requiert non seulement l'intégration des schémas ou modèles, mais aussi l'intégration des langages d'ontologies sous-jacents et même des logiques d'ontologies sous-jacentes.

Notre travail porte sur l'intégration des différentes approches de modélisation à base ontologique. Nous nous appuierons sur des standards PLIB et OWL qui résultent respectivement des efforts pour aboutir à des modèles d'ontologies de référence dans le domaine technique industriel et dans le domaine du WEB sémantique.

L'objectif de ce stage est tout d'abord d'étudier les modèles d'ontologies PLIB et OWL, d'identifier leurs différentes constructions puis, d'élaborer des règles permettant l'intégration dans

le modèle PLIB et au sein de l'architecture OntoDB[1] du modèle d'ontologie OWL, et aussi, l'intégration du modèle PLIB dans le modèle OWL.

Le travail présenté dans ce document est organisé comme suit :

- Dans la première partie, nous introduisons la notion d'ontologie et nous y présentons quelques problèmes qu'elle permet de résoudre et nous passons en revu les deux courants principaux de modélisation à base ontologique ; par l'étude des modèles PLIB ET OWL.

- Dans la seconde partie, nous étudions les différentes solutions d'intégration de ces approches et discutons des modifications à apporter à chaque modèle pour faciliter l'intégration.

Nous terminons par une conclusion générale dans laquelle nous présentons la démarche que nous avons adoptée par rapport aux résultats obtenus dans cette étude.

[1] OntoDB : nouvelle architecture de base de données proposée par le LISI ; dans cette architecture, les données et les ontologies qui en précisent le sens sont stockées dans une même base de données.

PARTIE I : PRESENTATION GENERALE

Introduction

Le premier chapitre de cette partie est consacré à la notion d'ontologie, nous y présentons les apports des ontologies et, leurs principaux domaines d'application. Le deuxième et le troisième chapitres sont respectivement consacrés à l'étude de PLIB ET OWL. Dans le but de faciliter les notions utilisées, nous évoquons brièvement dans le quatrième chapitre les techniques permettant l'échange de données entre différents modèles.

CHAPITRE I : LES ONTOLOGIES EN INGENIERIE DES DONNES.

I.1 La notion d'ontologie

I.1.1 Définition

La notion d'ontologie est apparue en informatique dans les années 90, une ***ontologie*** vise à décrire de façon consensuelle et partagée par les experts d'un domaine d'application assez large, l'ensemble des informations permettant de conceptualiser la connaissance de ce domaine. Ainsi, dans les domaines où il existe des ontologies, il est possible d'exprimer les différents modèles conceptuels correspondants aux besoins de diverses applications en termes de sous-ensemble ou de spécialisation d'une ontologie existante [PIE 01]. Plusieurs définitions ont été proposées pour cette notion ; la plus couramment citée est celle de Gruber [Gruber93] qui définit une **ontologie** comme *« an explicit specification of a conceptualization »*. Nous adoptons quant à nous la définition suivante : une **ontologie** est *« une spécification formelle consensuelle et référençable d'une conceptualisation d'un domaine »*.

- Le terme « c*onceptualisation* » fait référence à un modèle (au sens d'ensemble structuré) de concepts.

- Le terme « *formelle* » signifie que l'ontologie est représentée sous une forme traitable en machine et que cette description permet de rendre automatique certains traitements.

- Le terme « *consensuelle* » signifie que l'ensemble des membres d'une communauté se sont mis d'accord sur les concepts définis dans l'ontologie. Ces concepts pourront donc être utilisés pour se comprendre.

- Le terme « *référençable* » signifie que les différents concepts définis dans l'ontologie peuvent être référencés à partir de tout type d'application indépendamment du modèle de définition de l'ontologie.

I.1.2 Structure d'une ontologie

Une ontologie contient :
- Un ensemble de concepts (entités, attributs, …) représentatif du domaine couvert par l'ontologie ; cet ensemble est généralement représenté par une taxonomie,
- Les définitions des concepts et les relations entre les concepts (conceptualisation).
- Des assertions ou contraintes sur les concepts,
- Des instances qui sont des représentations des individus des concepts de l'ontologie.

Une ontologie a pour but d'être utilisée entre autres par des personnes, des applications et des bases de données qui souhaitent échanger les informations dans un domaine donné.

Pour être utilisée par les machines, elle doit :

- Avoir des définitions précises et une sémantique formelle
- Etre capable de s'adapter à l'usage courant à la fois de l'homme et des utilisateurs informatiques, et refléter les évolutions du modèle réel qu'elle décrit.

I.1.3 Classification des ontologies

Les concepts d'un domaine sont associés à un vocabulaire de termes pour chaque langue naturelle. En fait, certaines ontologies que nous qualifions de terminologiques ou linguistiques visent à définir des mots dans une langue donnée plus que des concepts. Cette constatation permet de classer les ontologies en deux principales catégories [PIE 03] :

1. Les ontologies linguistiques qui définissent non pas seulement des mots, mais également les relations qui les unissent et les règles qui leurs sont associées, telles que la synonymie ou l'antinomie. Les termes peuvent être introduits comme des concepts primitifs ou définis. Le nombre de ces termes définis est généralement très important. Un exemple très connu d'une telle ontologie est WordNet (http://wordnet.princeton.edu/perl/webwn).

2. Les ontologies conceptuelles visent à définir des concepts indépendamment d'une langue particulière. Le domaine d'étude est appréhendé au travers de concepts représentés par des classes et des propriétés. Des mots d'un langage naturel peuvent être associés mais ils ne définissent pas le sens des concepts. C'est l'ensemble des caractéristiques associées à un concept ainsi que ses liens avec les autres concepts qui en définissent le sens. Les ontologies conceptuelles peuvent ne définir que des concepts primitifs Elles permettent également décrire les équivalences conceptuelles.

I.2 Quelques domaines d'application des ontologies

I.2.1 Les bases de données

L'approche traditionnelle de modélisation de bases de données s'articule autour de trois niveaux :

1. le modèle conceptuel qui représente sous forme d'un modèle l'ensemble des concepts d'un domaine, ainsi que les liens qui les unissent,

2. le modèle logique qui est le résultat de conversions plus ou moins manuelles du modèle conceptuel. C'est une représentation du système tel qu'il sera implémenté en machine. Il consiste à décrire la structure de données utilisée sans faire référence à un langage de programmation,

3. le modèle physique qui est une description de la structure effective des données telles qu'elles existeront dans un SGBD particulier. Il est en fait la résultante du modèle logique après une opération de compilation.

Trois limites majeures ont été identifiées dans cette méthode de conception de base de données [Hondjack 02] :

1. L'insuffisance du pouvoir d'expression du modèle relationnel qui impose d'éclater un modèle conceptuel en tables de bas niveau.

2. La distance entre le modèle conceptuel (qui définit et décrit les concepts) et le modèle logique (qui décrit les données) : seul le modèle conceptuel est signifiant pour un utilisateur humain.

3. La forte dépendance entre le point de vue du modeleur et le modèle conceptuel résultant.

Le modèle conceptuel représente le centre du développement d'applications et il est ensuite traduit en un modèle logique (manière dont les informations seront représentées en machine). Une fois cette transformation effectuée, le modèle conceptuel est écarté du processus de développement ce qui entraîne la perte de la sémantique portée par ce modèle dans l'application qui en résulte. Il s'avère donc nécessaire de pérenniser le modèle conceptuel. Cela permettra de faire abstraction du modèle logique qui n'est pas forcément compréhensible et par ailleurs d'assurer une bonne évolution des modèles conceptuels. La pérennisation du modèle conceptuel nécessite de :

- Trouver une représentation des modèles conceptuels dans la base de données pour qu'ils puissent être facilement accessibles.
- Etablir une liaison entre le modèle logique et le modèle conceptuel présent dans la base de données.

Concernant le premier problème il faudrait, en plus de chaque modèle conceptuel, représenter de façon explicite dans la base de données la «*signification*» de chaque entité, attribut et association du modèle conceptuel. C'est précisément ce que les ontologies vont permettre.

Le second problème est résolu par l'architecture OntoDB [Pierra et al. 04] en cours d'implémentation au sein du LISI. Celle-ci permet de représenter ontologies et données au sein d'une même base de données. Ainsi, un utilisateur peut accéder directement à la signification des données et des éléments du modèle conceptuel qui les structurent grâce à l'ontologie stockée.

I.2.2 Le WEB sémantique [Tim Berners-LeeO5]

Les ontologies de domaine sont actuellement au cœur de nombreuses applications de l'Ingénierie des Connaissances, en particulier le Web Sémantique, car elles ont pour objectif de supporter la gestion des connaissances et le raisonnement sur ces connaissances, dans une optique d'interopérabilité sémantique entre agents humains et/ou artificiels.

Le Web sémantique n'est pas un Web à part, mais une extension du Web courant, dans lequel on donne à une information un sens bien défini pour permettre aux ordinateurs et aux gens de travailler en coopération, ceci dans le but de faciliter des opérations aussi variées que les courses en ligne, les démarches administratives, la formation en ligne, ou bien d'autres domaines.

Pour que le Web sémantique fonctionne, les machines doivent être capables de traiter et "comprendre" les données, avoir accès à des collections structurées d'informations et d'ensembles de règles d'inférence qu'ils peuvent utiliser pour parvenir à un raisonnement automatisé. Les systèmes traditionnels de représentation de la connaissance sont centralisés et nécessitent que chacun partage exactement une définition identique des concepts communs comme "parent" ou "véhicule". Ce contrôle central est étouffant et l'augmentation de la taille et de la portée d'un tel

système devient rapidement ingérable. Les réponses aux requêtes des utilisateurs sont des combinaisons de réponses obtenues après interrogation de plusieurs sources d'information. Un programme qui veut comparer ou combiner l'information provenant de deux bases de données doit savoir que ces deux termes sont utilisés pour désigner la même chose. Idéalement, le programme doit pouvoir découvrir des sens identiques pour n'importe quelle base de données rencontrée.

La solution à ces problèmes (terminologie, expression de sens, représentation de la connaissance, collecte intelligente d'information) se trouve dans les ontologies qui représentent une des principales visions du Web sémantique.

Avec les pages d'ontologies sur le Web, la signification des termes ou codes XML utilisés sur une page WEB peut être définie par des pointeurs de la page vers une ontologie dans ce domaine (WEB sémantique).

Les ontologies :

- Permettent l'intégration de sources d'informations
  - L'expression du sens.
  - La représentation de la connaissance.
  - La simulation de l'interrogation d'un système centralisé : l'utilisateur n'a pas accès aux données des sources. Il peut en ignorer le contenu. Il dialogue avec le système dans le vocabulaire de l'ontologie.

- Facilitent la communication avec le système de recherche d'informations
  - Aide à la formulation des requêtes
  - Aide à la saisie des requêtes grâce à une interface basée sur l'ontologie du domaine

- Aident à la formulation des requêtes
  - Développement d'outils graphiques de visualisation d'ontologies du domaine
  - Visualisation possible des définitions de concepts sous différentes formes : en (pseudo)français, sous forme normale ou étendue.
  - Classification automatique des concepts et visualisation de la hiérarchie des concepts résultante
  - Aident à la saisie des requêtes grâce à une interface basée sur l'ontologie du domaine : l'objectif est de guider l'utilisateur dans la construction de sa requête. Ce guidage est effectué par proposition de requêtes prédéfinies que l'utilisateur peut affiner.

Le pouvoir véritable du Web sémantique sera atteint quand les gens créeront de nombreux programmes ou agents qui collecteront les contenus du Web à partir de sources diverses, qui s'adapteront à l'évolution rapide du contenu présent sur Internet, qui traiteront l'information et échangeront les résultats avec d'autres programmes. L'efficacité de ces agents logiciels croîtra de manière exponentielle au fur et à mesure que seront disponibles des contenus du Web lisibles par des machines et des services automatisés (comprenant d'autres agents). Le Web sémantique promeut cette synergie : même les agents qui ne sont pas expressément fabriqués pour travailler ensemble peuvent transférer des données entre eux si les données sont accompagnées de la description de leur sémantique.

I.2.3 L'intégration de sources de données hétérogènes

La recherche de la bonne information à un moment précis est problématique et fastidieuse, nécessitant le plus souvent la connaissance par l'utilisateur de la structure de la base de données. Même lorsqu'on réussit à obtenir cette information, elle est difficilement exploitable et interprétable du fait de son caractère technique ou textuel.

Les bases de données à base ontologique représentent explicitement, non seulement les données (i.e. le modèle logique) mais aussi :

- L'ontologie elle-même, c'est à dire le sens des données,
- le modèle conceptuel, c'est à dire la structure des données représentées, et,
- le lien entre données et concept.

Ceci conduit à pouvoir accéder aux données au niveau connaissance. De plus, une ontologie ne fait que *décrire* les concepts qui *existent* dans le domaine cible. Malgré des points de vue applicatifs éventuellement différents, il est donc possible d'obtenir un consensus de tout un ensemble de concepteurs sur une ontologie. En fonction de son point de vue particulier, chaque concepteur pourra alors ensuite extraire de cette ontologie le sous-ensemble pertinent pour son point de vue. Ainsi :

- les modèles conceptuels pourront rester différents, mais
- les concepts communs aux deux modèles seront clairement identifiés : ils référencent le même concept de l'ontologie.

Ainsi, l'utilisation d'une ontologie apporte une solution aux différents problèmes posés par l'intégration de données (conflits de noms, conflits de représentation,…) [Bellatreche et al03]. L'approche de modélisation utilisant les ontologies ajoute un niveau supplémentaire (niveau ontologique). Une base de données à base ontologique permettra donc l'accès aux données au niveau signification, c'est-à-dire au niveau connaissance. Les ontologies constituent donc une référence pour la communication et l'échange d'informations entre les agents logiciels intelligents [Dieng 06] et apportent également une solution aux problèmes d'indexation et de recherche d'information notamment dans le cadre de la recherche d'information sur le WEB.

I.2.4 Le traitement du langage naturel

La reconnaissance de similitudes conceptuelles entre des mots permet essentiellement d'améliorer la recherche documentaire. Dans les moteurs de recherche actuels, une requête est composée d'un ensemble de mots éventuellement connectés par les opérateurs logiques OU, ET et NON. Le moteur produira sa réponse en fonction des mots contenus dans les documents parcourus. L'utilisation d'ontologies par ces moteurs doit permettre de retourner, en plus, les documents pertinents par rapport à la sémantique des mots de la requête.

I.3 Pourquoi intégrer les approches de modélisation à base d'ontologie ?

Les différentes approches de modélisation à base ontologique répondent chacune aux besoins d'un problème précis et ont été influencées par des courants différents.

Le langage OWL, influencé par les logiques de descriptions a pour principal objectif de permettre le raisonnement sur le contenu du WEB. Les ontologies basées sur OWL bénéficient ainsi de la puissance de ces logiques et principalement supportent des inférences sur leurs données. Le modèle OWL comporte des opérateurs permettant de définir des équivalences conceptuelles, utilisées ensuite pour calculer automatiquement des relations (raisonner) entre classes, propriétés et instances. Ainsi, dans les ontologies OWL la même information peut être représentée de différentes manières.

Le modèle PLIB a été influencé par les bases de données. Les ontologies PLIB offrent de bonnes performances au niveau stockage de données, permettent de représenter tout concept de manière unique, et adhèrent à l'hypothèse du monde fermé : ce qui n'est pas déclaré comme vrai est faux. Elles permettent également, contrairement aux ontologies de types OWL, de spécifier des contraintes d'intégrité sur les données et sont donc adaptées pour modéliser le contenu des bases de données.

Ainsi, chaque approche de modélisation possède des forces qui lui sont propres (degré d'expressivité, optimisation et capacité stockage de données, …), à cela, on peut ajouter le nombre et l'accessibilité des outils disponibles. Cependant, certains opérateurs PLIB, à l'exemple des expressions de contraintes, peuvent s'avérer nécessaires pour certaines ontologies OWL. Réciproquement, les opérateurs booléens pourraient s'avérer nécessaires pour spécifier des relations formelles entre différents schémas PLIB dans l'optique d'une intégration.

Conclusion

Nous avons présenté dans ce chapitre la notion d'ontologie. Nous avons fait un tour des différents domaines dans lesquels elles sont utilisées ainsi que leurs apports dans ces domaines. Nous avons brièvement présenté les approches existantes pour modéliser les données par les ontologies ce qui nous a permis de constater que les modèles existants sont plus ou moins adaptés selon le domaine à modéliser. Ce qui nous amène dans la suite à élaborer des règles qui vont permettre l'intégration de ces deux approches en essayant de garder la sémantique portée par chacune d'elles.

Pour cet objectif d'intégration, nous allons tout d'abord étudier comment ces modèles permettent de concevoir des ontologies. Ainsi, notre problématique dans l'étude de chaque modèle d'ontologies se résume aux questions suivantes:

- quels types d'ontologies le modèle permet-il de concevoir en terme de contenu et d'utilisation?
- quels sont les constructeurs du modèle et à quelle dimension de la connaissance sont-ils associés?
- comment sont définies la syntaxe et la sémantique de ces constructeurs?
- avec quelle précision le modèle permet-il de définir les concepts?
- comment le modèle permet-il la réutilisation et le partage des ontologies conçues?
- quels sont les mécanismes qui permettent de s'assurer de la cohérence des ontologies conçues?

Les chapitres 2 et 3 apporteront des réponses à ces questions pour les modèles d'ontologies PLIB et OWL respectivement.

CHAPITRE II : LE MODELE D'ONTOLOGIE PLIB

Nous présentons dans ce chapitre le modèle PLIB (Parts LIBrary : norme ISO 13584) conçu initialement dans le cadre du domaine technique mais qui est aujourd'hui utilisé dans bien d'autres domaines en ingénierie.

II.1 Les caractéristiques d'une ontologie PLIB

Une ontologie PLIB possède les caractéristiques suivantes [PIE02b]. Elle est :

- **conceptuelle :** chaque entrée (classe, propriété, domaine de valeurs ou source d'information) est un concept unique et complètement défini. Les mots qui apparaissent dans sa description n'en font que préciser le sens ;

- **multilingue :** chaque entrée est associée à un code qui constitue un identifiant universel permettant de désigner le concept correspondant. Les aspects textuels de la description peuvent apparaître dans un nombre quelconque de langues ;

- **formelle :** l'ontologie est définie dans le langage formel de spécification EXPRESS. Celui-ci étant compilable, les définitions de concepts, les référencements à des concepts et les contraintes d'intégrité que doivent respecter les instances des concepts sont traitables par machine.

- **modulaire :** Une ontologie peut référencer une autre ontologie pour en importer des catégories et/ou des propriétés sans avoir besoin de les dupliquer ;

- **multi-représentation :** Une fois défini, un concept peut être associé à un nombre illimité de représentations. Le point de vue qui caractérise chaque représentation est également un concept représentable en machine ;

- **consensuelle :** Le modèle conceptuel des ontologies PLIB a fait l'objet d'un consensus international et est publié sous forme de normes ISO et CEI[2]. Les ontologies conformes à ce modèle sont toutes développées soit à travers une démarche normative qui exige un consensus international sur le contenu, soit par des consortiums industriels regroupant un grand nombre de partenaires.

II.2 Les constructeurs du modèle PLIB [PIE 02]

Du point de vue de PLIB, une ontologie a pour but de définir, mais de la façon la plus précise possible, les catégories et les propriétés qui caractérisent les objets d'un domaine du monde réel, ainsi que les abstractions que les différentes communautés peuvent en construire [PIE02b]. Une ontologie est donc une collection de descriptions explicites, complètes et consensuelles de concepts :

dans le contexte le plus large où ces concepts ont un sens précis, et

2 CEI : Commission Electrotechnique Internationale (IEC : International Electrotechnical Committee)

sans aucune restriction ou règle correspondant à une utilisation particulière.

Cinq types de concepts peuvent y être définis :
1. les classes (catégories d'objets modélisés) : une classe est une collection d'objets définie en intention;
2. les propriétés (attributs des objets modélisés) : une propriété est une relation binaire entre deux classes ou entre une classe et un domaine de valeurs (le terme "attribut" est utilisé pour les méta-descripteurs de classes et de propriétés);
3. les domaines de valeurs (ou types) : un domaine de valeurs est un ensemble mathématique défini en extension ou en intention ;
4. les instances : une instance représente un objet appartenant à une classe. Toute définition ontologique émane d'une certaine source qui en assume la responsabilité.
5. les sources d'information (entité qui définit une ontologie ou une donnée).

L'objectif de précision signifie que face à un objet matériel ou un artefact donné appartenant au domaine ciblé par une ontologie, un utilisateur humain de l'ontologie doit savoir décider :

à quelles catégories l'objet appartient ou n'appartient pas,

quelles propriétés s'appliquent à l'objet, et

quelles grandeurs ou valeurs caractéristiques correspondent à chaque propriété applicable.

II.2.1 Identification des concepts

Afin de pouvoir référencer de façon non ambiguë et multilingue n'importe lequel de ces concepts, PLIB comporte un schéma d'identification universel (GUI: "Globally Unique Identifier"). Chaque source potentielle est associée à un identificateur unique (en général préexistant pour toute organisation ou établissement, par exemple en France il est construit sur différents codes SIRET ou SIRENE). Chaque source doit alors attribuer un code unique à chacune des classes qu'elle définit. Enfin le code d'une propriété doit être unique pour une classe et toutes ses sous-classes. La concaténation de ces codes permet alors d'identifier de façon unique et universelle chacun des concepts ci-dessus. C'est ce simple code, appelé un BSU (Basic Semantic Unit), qu'il sera suffisant de référencer pour caractériser une classe ou une propriété.

II.2.2 Les constructeurs pour définir les concepts

Chaque concept d'une ontologie PLIB est définit par trois éléments :
1. une unité sémantique atomique (basic_semantic_unit) qui permet une identification unique de ce concept;
2. sa définition (dictionary_element) qui décrit ce concept par un ensemble de propriétés;
3. ses instances (content_item).

Les classes sont organisées selon une hiérarchie (simple) avec factorisation/héritage des propriétés. Cette relation sémantique de subsomption est nommée is_a. Une autre relation sémantique nommée is_case_of permet également la subsomption entre classes. Celle-ci n'est cependant pas codée par le mécanisme d'héritage. Cette relation sémantique permet d'indiquer qu'une classe partage certaines propriétés avec une autre. Et, importe explicitement ces propriétés.

II.2.3 Les constructeurs pour définir les propriétés

Le modèle d'ontologie PLIB est basé sur deux principes fondamentaux.

1. La hiérarchie de classes et propriétés applicables. A chaque classe doivent être définies simultanément :
   - les propriétés applicables qui précisent la signification d'une classe ;
   - le domaine d'application qui précise la signification d'une propriété.

2. La distinction entre propriétés visibles, applicables et utilisées
   - Une propriété est définie au plus haut niveau de la hiérarchie où l'on peut la définir sans ambiguïté; elle est dite **visible** pour tout le sous-arbre correspondant.
   - Une propriété visible en un nœud peut y devenir **applicable**; cela signifie qu'elle est rigide [GAC2002] c'est à dire essentielle pour tout objet de la classe et pour toute sous-classe: toute instance doit présenter une valeur ou une grandeur qui représente cette propriété.
   - Enfin une propriété applicable peut ou non être **utilisée** dans la représentation d'une instance particulière de classe dans un univers formel particulier (base de données, échange informatisé,...).

Le co-domaine d'une propriété est défini par le constructeur *domain*. Ce peut être une classe ou un autre type de données.

II.2.4 Le système de type

Le schéma du modèle d'ontologie PLIB fournit un système de types permettant de représenter :

- des types de données primitifs tels que les entiers ou les réels ;
- des types de données énumérés (non_quantitative_code_type) ;
- des records (feature_class) ;
- des agrégats.

Il permet également d'associer plusieurs caractéristiques scientifiques comme les unités ou monnaies aux types de données. Par exemple, il permet en particulier de spécifier que la température du Futuroscope est fournie en degré Celsius. L'ensemble de ces types de données est présenté à l'annexe 6. Ce schéma montre notamment que le type de données permettant de définir le co-domaine d'une propriété comme une classe est class_instance_type.

II.2.5 Définition des instances d'un concept

PLIB propose deux approches pour définir les instances d'une classe.

1. Une approche explicite (en extension) qui consiste simplement à en énumérer les instances. Chacune est caractérisée par les valeurs prises pour les propriétés définies sur sa classe.

2. Une approche implicite qui consiste à décrire en intention et non en extension les ensembles d'instances. Un ensemble de contraintes permet de caractériser les instances licites d'un modèle. Cette description est réalisée par la définition d'un modèle de contraintes représenté dans une entité (model_class_extension).

II.3 Gestion du contexte dans le modèle PLIB

Permettre la définition et l'échange de catalogues de composants industriels a rapidement posé le problème de l'intégration de telles sources de données. Cette problématique a mis en évidence l'importance de la représentation du contexte de définition des concepts d'une ontologie. Pour la résoudre, G. Pierra [PIE 03] a identifié deux éléments de contexte qu'une ontologie doit représenter :

- le contexte de modélisation dans lequel chaque classe ou propriété est définie; par exemple, la largeur d'un moteur n'a de sens que si celle-ci est définie comme étant la largeur de sa boite englobante;
- le contexte d'évaluation d'une valeur; par exemple, le poids d'une personne dépend de la date à laquelle la mesure est faite; une valeur peut aussi dépendre d'une unité ou d'une monnaie à expliciter.

II.3.1 La modélisation des concepts

Le couplage fort imposé entre classes et propriétés (cf. section II.3.3) contribue à la modélisation de leurs contextes. En complément, le modèle d'ontologie PLIB propose de distinguer les propriétés essentielles d'une classe de celles qui dépendent d'un point de vue sur le concept représenté. Cette distinction est mise en place dans le modèle PLIB via trois catégories de classes :

- les classes de définition (component_class) qui contiennent les propriétés essentielles d'une classe;
- les classes de représentation (functional_model_class) qui contiennent les propriétés qui n'ont de sens que par rapport à un point de vue;
- les classes de vue (functional_view_class) qui définissent la perspective dans laquelle sont définies les propriétés des classes de représentation.

II.3.2 L'évaluation des concepts

Pour évaluer les valeurs de concepts, le modèle d'ontologies PLIB propose deux mécanismes :

1. Le premier permet d'associer à toute propriété, qui représente une quantité mesurable, une unité définie par un schéma EXPRESS standard (ISO 10303-41:2000).

2. Le second propose de distinguer les propriétés en fonction de leurs dépendances vis-à-vis d'autres paramètres. Le modèle PLIB définit ainsi quatre catégories de propriétés :
    - non_dependent_P_DET : les propriétés essentielles d'une classe; elles sont indépendantes des autres propriétés;
    - dependent_P_DET : les propriétés dont la valeur dépend de paramètres de contexte;
    - condition_DET : les propriétés qui représentent des paramètres de contexte;

- representation_P_DET : les propriétés non essentielles et dont la valeur peut changer sans que cela ne change l'objet décrit; ce sont donc les propriétés de classes de vue ou représentation.

II.3.3 Les principes fondamentaux de la modélisation d'ontologies PLIB

Arriver à un consensus sur la définition d'une hiérarchie de classes PLIB est certainement l'une des principales difficultés de conception d'une ontologie. Le fait de ne pas pouvoir utiliser d'héritage multiple limite les conceptions possibles. Cependant, il existe souvent plusieurs critères pour décrire la spécialisation dans une arborescence.

Utiliser l'héritage multiple est dans certains cas une solution mais, ceci conduit vite à une explosion combinatoire des classes lorsque le nombre de critères augmente [Sardet 99]. Pour régler ce problème, sans utiliser l'héritage multiple, PLIB introduit deux notions :

1. le sélecteur de classes qui consiste à associer une propriété à une classe qui ne prend qu'une seule valeur pour toutes ses instances ;
2. le mécanisme is_case_of qui est une implémentation particulière de la subsumption dans lequel une classe indique le sous-ensemble des propriétés de sa classe subsumante qu'elle souhaite importer.

La méthodologie associée à PLIB préconise l'utilisation de la modélisation par propriétés, elle recommande que :

1. une classe ne soit introduite dans la hiérarchie que si elle constitue le domaine d'une nouvelle propriété qui n'aurait pas de sens dans les classes subsumantes. PLIB est orienté propriété.
2. le critère à privilégier pour concevoir une hiérarchie soit celui permettant la factorisation maximale des propriétés.
3. deux propriétés définies dans deux classes différentes possèdent la même sémantique si :
   - dans une circonstance particulière, les deux classes peuvent être interchangées et dans ce cas les deux propriétés possèdent la même valeur ou si,
   - les propriétés jouent un rôle identique lorsqu'un traitement est appliqué sur un ensemble d'instances pouvant appartenir à l'une des deux classes.

II.4 Modèle Formel d'une ontologie PLIB [PIE 05].

Formellement, une ontologie PLIB peut être définie comme un quadruplet :

O :< C, P, Sub, Applic>, avec :

- C : l'ensemble des classes utilisées pour décrire les concepts d'un domaine donné (comme les pannes des équipements, les composants électroniques, etc). Chaque classe est associée à un identifiant universel globalement unique (GUI).

- P : l'ensemble des propriétés utilisées pour décrire les instances de l'ensemble des classes C. Nous supposons que P définit toutes les propriétés susceptibles d'être présentes dans une

base de données. Chaque propriété est associée à un identifiant universel globalement unique (GUI).

- Sub : $C \rightarrow 2^C$ est la relation de subsomption1 (*is-a* et *is-case-of*) qui, à chaque classe de l'ontologie, associe ses classes subsumées directes. Sub définit un ordre partiel sur C.

- Applic : $C \rightarrow 2^P$, associe à chaque classe de l'ontologie les propriétés qui sont applicables pour chaque instance de cette classe. Les propriétés qui sont applicables sont héritées à travers la relation *is-a* et peuvent être importées de façon explicite à travers la relation de *case-of*.

Une ontologie PLIB n'est pas un modèle conceptuel. Le fait qu'une propriété soit applicable pour une classe signifie qu'elle est rigide, c'est-à-dire que chaque instance de cette classe appartenant à l'univers du discours devra posséder une caractéristique correspondant à cette propriété. Cela ne signifie pas que la valeur d'une telle propriété devra être explicitement représentée pour chaque représentation d'une telle instance dans la base de données.

Le schéma EXPRESS-G du modèle PLIB est présenté en annexes (cf. annexes 1 à 6).

II.5 OntoDB : un modèle d'architecture pour les bases de données à base ontologiques.

Un modèle d'architecture appelé OntoDB a été proposé pour les bases de données à base ontologique. Cette architecture est actuellement en développement au sein du LISI.

II.5.1 Présentation

L'architecture OntoDB est constituée de quatre parties. Cette architecture comporte en plus des deux parties traditionnelles de tout SGBD (méta-base et modèle logique) deux parties supplémentaires (méta-schéma et ontologie) spécifiques à OntoDB comme le montre la figure2.1 :

1. La partie *ontologie* permet de représenter complètement les différentes ontologies des domaines couverts par la base de données, ainsi que les liens de celles-ci avec de possibles ontologies externes et les sous-ensembles d'ontologie qui définissent le modèle conceptuel de la base de données.

2. La partie *méta-schéma* permet de représenter, au sein d'un modèle réflexif, à la fois le modèle d'ontologie utilisé et le méta-schéma lui-même (cette partie permet de rendre générique tout traitement sur les ontologies). Pour la partie ontologique, le méta schéma joue le même rôle que celui de la méta-base d'une base de données traditionnelle. Il permet d'avoir un accès générique à la partie ontologie et permet de stocker plusieurs modèles d'ontologie, par exemple PLIB et OWL.

3. Une partie *méta-base ou catalogue système* permet de représenter le schéma logique de représentation des objets du domaine couvert par les ontologies. Toutes les tables des autres parties sont documentées dans la méta-base. Cette partie représente aussi le lien du modèle logique des données avec l'ontologie.

4. Une partie *données* représentant les objets du domaine; elle représente les instances des classes d'ontologie, toute donnée étant liée à un élément particulier de l'ontologie qui en définit le sens. C'est la partie "données" d'une base de données traditionnelle.

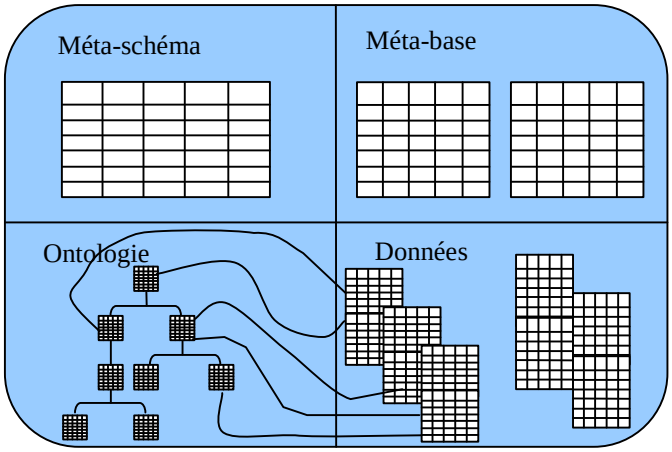


Figure 2.1 : Le modèle d'architecture OntoDB

II.5.2 Extensions prévues d'OntoDB

Les extensions prévues d'OntoDB concernent notamment l'expression :

- des contraintes structurelles : unicité de clé, contraintes référentielles (clés étrangères), des dépendances fonctionnelles et contraintes de domaine ;
- des contraintes de comportement : essentiellement des dérivations fonctionnelles qui consistent à fournir une fonction pour calculer la valeur d'une propriété.

L'extension du pouvoir d'expression de PLIB par ces ajouts permettra, en plus de redéfinir le domaine d'une propriété, d'indiquer que deux propriétés sont l'inverse l'une de l'autre et d'exprimer explicitement le lien entre paramètres de contexte et propriétés dépendantes de ces paramètres. L'expression de dépendances fonctionnelles permet également une nouvelle forme de subsomption [Ait-Ameur 04] qui est un mécanisme essentiel notamment pour l'intégration de sources de données hétérogènes.

Le chapitre suivant montre que d'autres mécanismes sont fournis par les langages du Web Sémantique, et plus généralement dans les logiques de description, pour permettre l'intégration de sources hétérogènes. Comme nous le verrons, l'ajout de tels mécanismes dans le modèle PLIB est actuellement à l'étude.

CHAPITRE III : LE LANGAGE D'ONTOLOGIE OWL

Initialement destiné à être lu, presque tout le contenu du Web (lisible par les machines mais pas toujours compréhensible par celles-ci) est aujourd'hui destiné à être manipulé de façon intelligente par des programmes informatiques [Tim Berners-Lee 99].

La vision du Web sémantique est de permettre d'automatiser l'inter-opération entres les entités sur le Web. Une telle interopérabilité peut être atteinte en annotant le contenu du Web avec les termes ontologiques traitables en machine. Dans ce cas, une ontologie consiste typiquement en un certains nombres de classes, des relations entre les classes, des individus et des axiomes. Ces éléments sont exprimés en utilisant certains langages logiques.

Le défi du Web sémantique, a été de fournir un langage qui exprime à la fois des données et des règles pour raisonner sur les données et pour que les règles de n'importe quel système de représentation de la connaissance puissent être exportées sur le Web.

Le désir d'ajouter de la logique au Web (c'est-à-dire lui donner la possibilité d'utiliser les règles pour faire des inférences, choisir des cours d'action et répondre aux questions) a conduit à la naissance de différentes approches pour traiter les données. En particulier, les modèles d'ontologies pour concevoir des ontologies WEB sont issus d'approches basées sur la logique des descriptions.

Exprimer de la connaissance sur le Web est l'ambition du Web sémantique. Au-delà de ce simple mot d'ordre, diffuser des ontologies sur le Web est le moyen de permettre à d'autres de se les approprier, de les étendre et de les réutiliser. Mais si les ontologies doivent s'échanger librement sur le Web, il est nécessaire de les intégrer plus aux langages du Web. Pour cela, le W3C a développé deux langages d'ontologies compatible avec RDF et permettant de le contraindre; le premier est RDFS qui est en fait un langage simplifié, le second est OWL qui est un langage d'expression d'ontologies plus expressif basé sur les logiques de descriptions.

Le standard OWL s'ajuste dans la vision du WEB à une pile de langages incluant XML, RDF et RDFS pour ne citer que les plus importants. De ce fait, OWL maintient autant que possible la compatibilité avec ses prédécesseurs. OWL intègre donc à la fois la facilité d'énoncer de RDF, la capacité de structuration de RDF Schema et les étend de manière importante en se basant sur le paradigme des frames et les logiques de descriptions pour la spécification formelle et la structuration de sa syntaxe abstraite.

Ce chapitre est consacré à l'étude de OWL, nous allons tous d'abord commencer par l'étude de ses prédécesseurs RDF et RDF Schema.

III.1 RDF

RDF ("Resource Description Framework") est le premier langage apparu pour définir la sémantique de sources WEB.

RDF permet de décrire tout élément selon un mécanisme particulièrement simple : il s'agit de phrases minimales, composées d'un sujet, d'un verbe et d'un complément ; on parle de *déclaration*

RDF. Par exemple : « *Chimène est étudiante en Master2* » est une déclaration RDF possible. On remarquera la proximité de RDF avec le langage naturel.

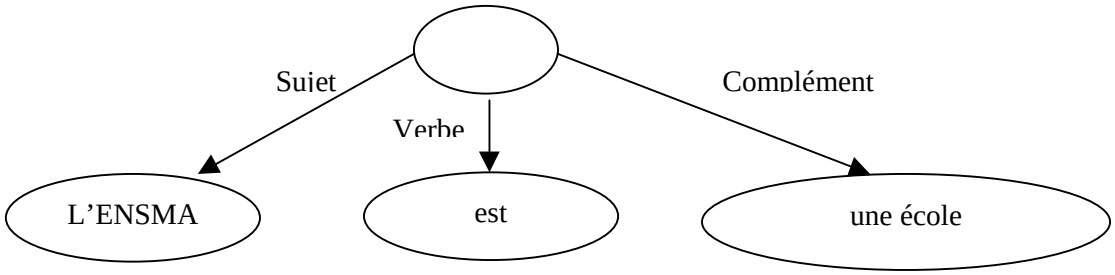


Figure 3.1 : Exemple de déclaration RDF

La vocation initiale de RDF est l'utilisation de méta-données pour décrire les ressources du Web, le terme "ressource" représentant toute information pouvant être référencée par un URI (Uniform Resource Identifier), comme par exemple un site Web complet, une page Web ou même un élément particulier de cette page Web. L'idée est donc de rajouter une dimension descriptive aux ressources véhiculées sur le Web, cela en leur adjoignant des descriptifs explicites qui les rendront interprétables et utilisables par les outils du Web.

RDF ne peut se permettre la fantaisie d'omettre l'un des trois éléments de sa phrase. En français, par exemple, telle personne pourra exprimer en regardant le ciel : « il pleut » ; cette personne sous-entend naturellement qu'il pleut des gouttes d'eau à l'endroit où elle se situe. En RDF on exprimera alors explicitement toutes les composantes de la proposition : « Des gouttes d'eau [sujet] pleuvent [verbe] en ce lieu [complément] ». Ce formalisme peut, dans certains cas, paraître un peu lourd, mais il est nécessaire pour analyser des propositions automatiquement et sans ambiguïté. Grâce à RDF, il devient possible d'écrire des programmes informatiques capables d'opérer des traitements sur des connaissances : indexation, classement, diffusion, formatage, comparaison, inférence, etc.

III.1.1 Caractéristiques de RDF

Un modèle RDF est définit à partir de :

- un ensemble appelé ***ressources***.
- un ensemble appelé ***littéraux***.
- un sous-ensemble de *ressources* appelé ***propriétés***.
- un ensemble appelé ***déclarations***, dont chaque élément est un triplet de la forme : {pred, sub, obj} ; où pred est une propriété (membre de *propriétés*), sub est une ressource (membre de *ressources*), et obj est ou bien une ressource ou bien un littéral (membre de *littéraux*).

Autrement dit, RDF définit une ressource (URI) sous la forme d'un triplet : ressource, propriété, valeur. Formulé encore autrement, on peut dire qu'une chose est décrite sous la forme d'une phrase : sujet, verbe, complément. Ce modèle permet de représenter un nombre considérable d'éléments.

Nous reformulons cette définition comme suit : « toutes les « choses » décrites par des expressions RDF sont appelées des ressources. Ces ressources ne sont pas uniquement les

ressources du Web, mais peuvent être aussi un objet qui n'est pas directement accessible par le Web, par exemple « un livre imprimé ».

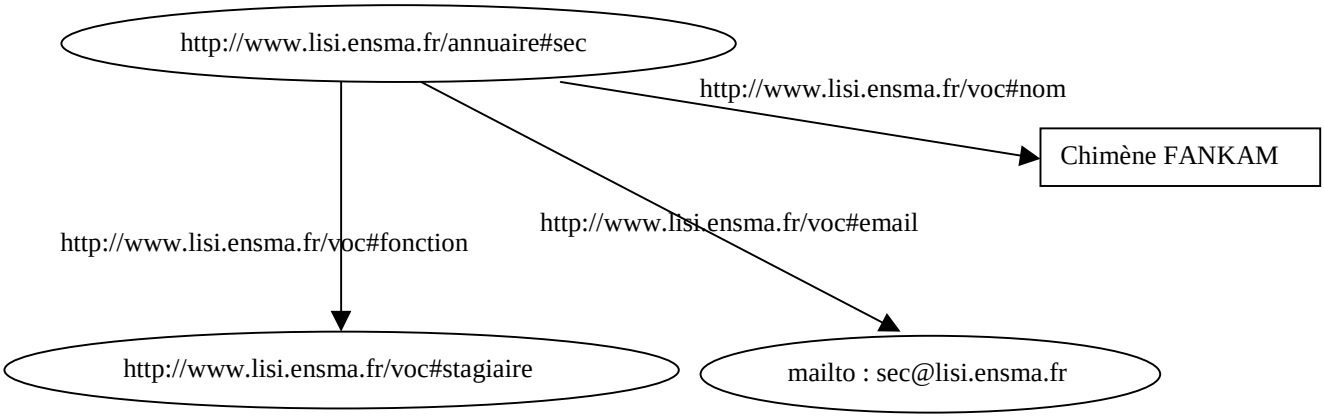
- Les *ressources* sont toujours nommées par des URIs avec des ancres ou IDs optionnelles. Toute ressource peut avoir une URI. L'extensibilité des URIs permet l'introduction d'identificateurs pour toute ressource imaginable.
- Une *propriété* est un aspect, une caractéristique, un attribut ou une relation spécifique utilisée pour décrire une ressource.
- Chaque propriété possède une signification spécifique, définit ses valeurs permises, les types de ressources qu'elle peut décrire, et les relations qu'elle entretient avec les autres propriétés.

Une ressource spécifique associée à une propriété, définie ainsi que la valeur de cette propriété pour cette ressource est une *déclaration* RDF.

Ces trois parties d'une déclaration sont appelées, respectivement, le *sujet*, le *prédicat*, et l'*objet*.
L'objet d'une déclaration (c.-à-d. la valeur de la propriété) peut être une autre ressource (spécifiée par une URI) ou il peut être littéral (une simple chaîne ou autre type de données primitif défini par XML). On peut donc imaginer les propriétés RDF comme les attributs de ressources et, en ce sens, elles correspondent aux paires traditionnelles « attribut, valeur ».

III.1.2 Le modèle formel de RDF

Un ensemble de descriptions RDF est un graphe orienté étiqueté dans lequel les nœuds sont des ressources (ou des littéraux) et les arcs représentent les propriétés des ressources. Un schéma RDF définit le vocabulaire des étiquettes des nœuds du graphe (appelées classes ou types de littéraux) et des arcs (appelés types de propriété). Les deux types d'étiquettes peuvent être organisées en ***taxonomie*** (hiérarchie de spécialisation).



- Ellipse = URI (sujet ou objet)
- Rectangle = Littéral (objet)
- Arc = Prédicat

Figure 3.2 : Exemple de graphe RDF

III.1.2.1 Définition d'un schéma RDF

Notons :
U : l'ensemble d'URI,
B : l'ensemble de nœuds blancs (ressource anonymes, ressource locale sans identificateur),
L : l'ensemble des littéraux.

Un graphe RDF est un ensemble $G = \{(v_1, v_2, v_3) \in (U \cup B) \times U \times (U \cup B \cup L)\}$.
Les éléments (v_1, v_2, v_3) de G sont appelés des déclarations. Une déclaration (v_1, v_2, v_3) signifie que le "sujet v_1 a comme valeur pour la propriété v_2 l'objet v_3".

III.1.3 De RDF à RDF Schéma : les insuffisances de RDF

RDF est un modèle de méta-données puissant, largement accepté et utilisé. C'est un format très ouvert et malléable dédié à l'expression d'assertions sur les relations entre objets, il ne permet cependant pas :

- de définir les propriétés des classes dont ses objets sont instances.
- d'identifier les termes sur lesquels une interrogation doit porter,
- de spécifier le fait que deux triplets doivent toujours apparaître ensemble car les triplets RDF sont indépendants,
- d'interdire les références circulaires ; tous les triplets RDF sont accessibles,
- de formuler des contraintes sémantiques plus riches et de faire des raisonnements; RDF permet de représenter des déclarations de propriétés sur des ressources, mais ne permet pas d'exprimer des connaissances sur les propriétés ou sur les types de ressources :
    - Quelles sont les propriétés autorisées sur un type de ressources ?
    - Quelles sont les valeurs autorisées pour une propriété ?
    - Quels sont les liens entre les types de ressources (généralisation / spécialisation) ?

III.2 RDFS

III.2.1 Présentation

RDF Schema fondé sur RDF permet de définir des vocabulaires[3]. C'est un des piliers du Web sémantique puisqu'il permet de bâtir des concepts, définis par rapport à d'autres concepts, ayant la particularité d'être partagés à travers le Web.

Grâce à RDF Schema, nous pouvons par exemple définir que le concept de « berger » dans le vocabulaire intitulé « professions », représente la profession de berger. Une fois que le vocabulaire « professions » est formellement défini grâce à RDF Schema, n'importe qui peut désormais utiliser la notion de « berger » sachant avec certitude qu'elle correspond à la profession de berger. Un outil intégrant le vocabulaire « professions » pourra par exemple proposer à ses utilisateurs de décrire leur profession, sur la base de la liste des professions définies par le vocabulaire « professions ». Les données de cet outil pourront être publiées sur Internet et faire l'objet d'une indexation par un autre outil connaissant le vocabulaire « professions » : les utilisateurs de ce dernier outil pourront donc faire la demande, par exemple, de lister toutes les

3 Vocabulaire : ensemble de termes utilisés pour étiqueter, décrire des choses.

personnes occupant la profession de berger. RDF Schema est un « système de typage » pour RDF.

Par extension, un schéma RDF désigne un vocabulaire défini avec la norme RDF Schema. On parle aussi de « vocabulaire RDF ». RDF Schema ne permet que de décrire des vocabulaires simples; pour des vocabulaires plus complexes, on se tournera vers OWL qui enrichit le modèle RDF Schema.

III.2.2 Caractéristiques de RDF Schema

RDF Schema propose un ensemble de ressources (ou méta-ressources) qui vont pouvoir être utilisées afin de définir des schémas RDF spécifiques (appelés vocabulaires RDF), les ressources définies dans ces derniers étant utilisées pour caractériser des ressources du Web. Notons que RDF Schema est lui-même un vocabulaire RDF : les ressources qu'il introduit sont définies en RDF. RDF Schema définit donc un paradigme de modélisation à partir duquel des objets du monde réel vont pouvoir être décrits. Ainsi, RDF Schema est doté du nombre minimum de concepts nécessaires à la définition d'un vocabulaire. RDFS définit :

- la notion de **classe** qui est un ensemble de plusieurs objets (ex : la classe des « bergers ») et une hiérarchie de spécialisation sur les classes.

- des **propriétés** et une hiérarchie de spécialisation sur les propriétés.

- des **restrictions** sur la valeur d'une propriété et sur le type de ressource décrit par la propriété.

- la propriété particulière « ***est une sous-classe de*** » qui permet de définir qu'une classe est un sous-ensemble d'une autre classe (ex : la classe « bergers » est sous-classe de la classe « professions »).

- la classe ***ressource*** classe mère de toute chose.
  - Tout est une ressource dans le Web sémantique, sauf la notion de littéral.
  - Toute classe est une sous-classe de la classe ressource.
  - Il définit la notion de ***littéral*** qui est une valeur comme une chaîne de caractère ou des chiffres : ces choses ne sont pas des concepts et ne peuvent être manipulés comme tels.

- la propriété ***range*** (« **s'applique à la classe** ») permettant de spécifier le champ d'application d'une propriété (ex : la propriété « sont gardés par » s'applique aux classes des « bergers » et des « nurses anglaises »).

- propriété ***domain*** (« **est l'objet de la propriété** ») permettant ainsi de spécifier quelles sont les classes auxquelles ont peut affecter telle ou telle propriété (ex : la classe des « moutons » peut être l'objet de la propriété « sont gardés par »).

III.2.3 Le modèle formel de RDFS

III.2.3.1 Définition d'un schéma RDFS

Notons C : l'ensemble des noms de classes,
P : l'ensemble des noms de propriétés,
L : l'ensemble des types de littéraux et,
< : la relation de subsumption.

Un schéma RDF est un quintuplet RS = (V_s, E_s, ψ, λ, H) où :

- V_s et E_s représentent respectivement l'ensemble des noms de nœuds et d'arcs ;
- ψ est une fonction de contrainte sur les propriétés : $\psi : E_s \rightarrow V_s \times V$, $\forall e \in E_s$, domain(e) $\in V_s \subset C$ et range(e) $\in V_s \subset C \cup L$. (la fonction de contrainte associe à chaque arc ou propriété un couple (domain, range) tel que « domain » est un élément de l'ensemble des classes et « range » est soit une classe soit un littéral).
- λ est une fonction de typage définie sur : $\lambda : V_s \cup E_s \rightarrow T$, T est l'ensemble des noms de types manipulés en RDF (type classe, type propriété, type littéral, type URI, type bag, type séquence, type Alternative) ;
- H représente la taxonomie notée : $H = (C \cup P, <)$

III.2.3.2 Définition d'une instance (statement) RDFS

Un ensemble d'instructions RDF, instances d'un schéma RDF, est un quintuplet RD = (V_D, E_D, ψ, ν, λ) où :

- V_D et E_D représentent respectivement l'ensemble des noms de nœuds et d'arcs ;
- ψ est une fonction de contrainte sur les propriétés : $\psi : E_D \rightarrow V_D \times V_D$;
- ν est une fonction de valuation : $\nu : E_D \rightarrow V$, V étant l'ensemble des valeurs associées à des instances de classes, propriétés, types littéraux, URI : v associe à chaque nœud *n* de V_D sa valeur
- λ est une fonction de typage définie sur : $\lambda : V_D \cup E_D \rightarrow 2^N \cup \{Seq, Bag, Alt\}$:
- à chaque noeud *n* de V_D, λ associe un ensemble de noms de classes de C (car une ressource peut être instance de plusieurs classes) ou le nom de l'un des types de containers autorisés en RDF (Bag, Seq, Alt),
- à chaque arc *e* de E_D, λ associe le nom de la propriété p de P associée.

III.2.4 De RDF Schema à OWL : les insuffisances de RDF Schéma

RDFS permet de représenter des prédicats binaires, les relations de spécialisation entre classes (respectivement entre propriétés) et des restrictions sur les domaines de valeur des prédicats binaires (range, domain). Cependant, l'expressivité de RDFS est trop limitée pour représenter la sémantique portée par les uses-cases identifiés par le W3C [Schreiber 02]. RDFS est trop faible pour décrire les ressources en détail :

- Aucune contrainte locale sur le domaine et le co-domaine d'une propriété : par exemple : la propriété *hasChild* est une personne quand on l'applique aux domaines des Personne, mais elle est un petit éléphant sur les domaine des Éléphant.
- Aucune contrainte sur l'existence/cardinalité : il est impossible d'exprimer que toutes les instances de Personne ont une mère qui est aussi une personne, ou bien que toute personne a exactement 2 parents.

- Pas d'exclusion entre classes (deux classes ne peuvent avoir d'instance commune)
- Pas de liens plus précis entre classes et entre propriétés (classes équivalentes, transitivité d'une relation, etc.).
- Son système de types est très limité (mais est extensible).
- La définition de ses relations de cardinalité est très élémentaire et il ne permet pas d'exprimer des contraintes ensemblistes.
- Il est difficile de fournir un support pour le raisonnement à RDF-S.

D'où la nécessité de disposer de connaissances ontologiques pour raisonner efficacement sur des connaissances factuelles exprimées en RDF.

Le WEB sémantique a besoin d'une ontologie pour définir les concepts et les relations qui vont décrire et représenter un domaine de connaissances, définir les terminologies utilisées dans un contexte particulier, les contraintes sur les propriétés, les caractéristiques logiques des propriétés, et l'équivalence des termes .

L'intérêt d'utiliser une ontologie est de pouvoir valider ses sources : par exemple, être sûr que l'éditeur d'un ouvrage est bien un Publisher et que son adresse est connue. Mais c'est surtout d'augmenter le nombre d'inférences possibles à partir des données. Plus précisément, une telle modélisation devrait permettre de répondre à des requêtes complexes utilisant les modèles pour compléter la connaissance disponible. Ainsi, plus on fait intervenir d'ontologies pertinentes dans l'évaluation d'une requête, plus les réponses sont pertinentes.
Disposer de telles ontologies, et de moteurs d'inférence associés, ouvre la voie à de nombreuses applications:

- La complétion de requêtes;
- La connexion de services par la mise en correspondance de leurs descriptions;
- La réponse à des requêtes impliquant des pages Web distribuées sur le Web.

III.3 OWL

III.3.1 Ambitions de OWL

Comme RDF, OWL est doté d'une sémantique en théorie des modèles (Cf. section III.1.2) permettant de spécifier tout ce qui est conséquence d'un ensemble d'assertions de OWL. Elle est directement issue des logiques de descriptions. La sémantique associée aux mots-clés de OWL est plus précise que celle associée aux documents RDF représentant une ontologie OWL

Ainsi, si l'on étudie la vie de Bertrand Russell, on voudra trouver les Biographies de tous les coauteurs et élèves de Bertrand Russell. Bien entendu une telle requête devra retourner les Biographies et les Autobiographies ; elle devra aussi trouver les personnes dont le doctorat a eu comme superviseur Bertrand Russell. Ces deux informations n'ont pas à être dans la requête, il suffit qu'elles soient présentes dans l'ontologie pour être exploitées.
L'information sur les élèves d'un auteur n'est typiquement pas celle que l'on trouve dans les notices bibliographiques actuelles mais que l'on peut trouver sur les pages Web des auteurs comme sur les sites qui leur sont consacrés. Leur encodage en OWL permettra de les exploiter.

Disposer d'un langage de description d'ontologies standard permet principalement de publier sur le Web des ontologies qui puissent être appréhendées par de nombreux acteurs.

Indépendamment de l'origine des données, elles pourront être interprétées en fonction d'une ontologie décrite en OWL.

III.3.2 Les influences qui ont guidé la conception d'OWL

La première et principale source d'influence de OWL est issue de ses prédécesseurs. Parmi les impératifs de conception de ce langage figure la compatibilité avec RDF. Celle-ci se traduit par l'utilisation de RDF/XML comme syntaxe concrète de OWL. Seulement, cette compatibilité ne doit pas être uniquement syntaxique mais aussi sémantique. Ce problème a été résolu en s'inspirant des solutions trouvées par d'autres prédécesseurs de OWL tels que SHOE, OIL et DAML+OIL [Jean 04].

Le domaine de la logique de description (Description Logics) est la seconde source d'influence de OWL. Elle a, d'une part, influencé le choix de spécifier la sémantique du langage par la théorie du modèle. D'autre part, les études sur la complexité menées dans ce domaine ont également orienté les choix des constructions du langage. En effet, un autre objectif important de OWL est de pouvoir assurer qu'il est possible de lui associer un moteur d'inférence décidable basé sur ce langage. Ceci signifie qu'il existe un algorithme pour évaluer la subsomption entre deux ontologies. Par abus de langage, dans la suite de ce mémoire, nous utiliserons le qualificatif décidable sur le langage OWL pour indiquer cette caractéristique.

Enfin, la troisième source d'influence est le domaine des Frames (Frames paradigm) qui consiste à regrouper l'ensemble des informations qui se rattachent à un même élément. Ceci a influencé la conception de la syntaxe abstraite du langage présentée dans la section III.3.5.

III.3.3 OWL : une variante des logiques de description

Les logiques de description (DLs), appelées aussi logiques terminologiques, sont une famille de formalismes conçue pour décrire les connaissances d'un domaine d'application par un moyen clair, formel et structuré, mais également de raisonner avec ces connaissances au travers de leurs descriptions. Elles représentent un fragment décidable des logiques de premier ordre, elles sont donc favorables au raisonnement automatisé.

D'autre part, les Logiques de Description diffèrent de leurs prédécesseurs, tels que Réseaux Sémantiques et Frames, étant donné qu'elles sont équipées d'une logique formelle basée sur des sémantiques formelles.

Les Logiques de Description (DLs) ou logiques terminologiques sont donc une famille de langages de la représentation de la connaissance qui peuvent être utilisées pour représenter la connaissance d'un domaine

III.3.3.1 Modélisation de la connaissance

Les connaissances du domaine sont représentées par des descriptions de deux types :

1. celle décrivant en intension des ensembles d'individus (à l'instar des classes d'objets). Ces descriptions sont formées à l'aide d'un ensemble d'opérateurs qui décrivent les individus en restreignant leurs propriétés. Ces descriptions constituent la première partie de la base de connaissances du domaine : c'est la partie

intensionnelle du domaine, appelée TBox (elle correspond au diagramme de classes en UML) ;

2. celle décrivant une instanciation de la partie intensionnelle. Elle exprime des assertions d'appartenance d'un individu à une classe ou d'une relation entre deux individus. Cette composante est appelée ABox (elle correspond au diagramme des instances en UML).

Les logiques de description permettent de représenter les connaissances relatives à un domaine de référence à l'aide de « **descriptions** » qui peuvent être des **concepts** ou classes, des **rôles** ou **propriétés** et des **individus** [Napoli 97].

Deux types de concepts sont rencontrés :

1. Les **concepts primitifs** où les rôles déterminent des conditions nécessaires d'appartenance à l'extension du concept.

2. Les **concepts définis** où les rôles déterminent des conditions nécessaires et suffisantes d'appartenance à l'extension du concept.

Les concepts modélisent des classes d'individus et les rôles des relations entre classes. Une sémantique est associée aux descriptions par l'intermédiaire d'une fonction d'interprétation :
Les concepts sont interprétés comme des sous-ensembles d'un domaine d'interprétation $\Delta^{\mathcal{I}}$ *et les rôles comme des sous-ensembles du produit* $\Delta^{\mathcal{I}} X \Delta^{\mathcal{I}}$ *. Pour un concept C,* $C^{\mathcal{I}}$ *correspond au sous-ensemble des éléments du domaine* $\Delta^{\mathcal{I}}$ *qui appartiennent à l'extension de C, et pour un rôle r,* $r^{\mathcal{I}}$ *correspond au sous-ensemble des couples d'éléments du produit* $\Delta^{\mathcal{I}} X \Delta^{\mathcal{I}}$ *qui appartiennent à l'extension de r.*

III.3.3.2 Les opérations à la base du raisonnement terminologique

Les logiques de description offrent un certain nombre d'opérations qui sont alors à la base du raisonnement sur les descriptions, ou raisonnement terminologique. Les plus importants sont :

- le test de subsomption, qui permet de tester si une description est plus générale qu'une autre, ce test permet d'organiser les concepts et les rôles en hiérarchies ;
- La classification permet de déterminer la position d'un concept et d'un rôle dans leurs hiérarchies respectives ;
- le test d'instanciation, qui permet de retrouver les concepts dont un individu est susceptible d'être une instance. ;
- le test de satisfiabilité d'un concept par rapport à une base qui permet de vérifier qu'un concept admet des instances ;
- le test de satisfiabilité d'une base qui permet de vérifier qu'une base admet un modèle ou des interprétations.

D'autres types de raisonnements sont apparus par la suite, appelés raisonnements non standards, comme la réécriture de concepts et la différence.

Il existe à ce jour plusieurs familles de logiques de description, que l'on distingue selon le type de raisonnement et les constructeurs qu'elles offrent. Toutes ces familles sont construites avec un accent sur la décidabilité des principaux problèmes de raisonnement et fournissent des services de raisonnement complets et vérifiables : formalisme de la sémantique, choix des constructeurs, intégration des types de données et des valeurs.

Les Systèmes de Logiques de Description fournissent à leurs utilisateurs des possibilités d'inférences variées qui déduisent la connaissance implicite de la connaissance représentée

explicitement. Le pouvoir d'expression d'OWL, fortement influencé par les différentes familles de logique de description est déterminé par les constructeurs de classes et de propriétés supportés et par les types d'axiomes qu'on peut avoir dans une ontologie.

III.3.3.3 Les constructeurs OWL issus des logiques de descriptions

Le pouvoir d'expression d'OWL est déterminé par les constructeurs de classes et de propriétés supportés et par les types d'axiomes qu'on peut avoir dans une ontologie. La conception de OWL a été principalement basée sur la famille de logique de description SH.

- SH : étend le langage minimal AL (cf. section III.3.3.5) en y ajoutant les propriétés transitives et la hiérarchie de propriétés.
- SHIQ : ajoute à SH l'inverse de propriété et les restrictions de cardinalité;
- SHOQ : ajoute à SH la possibilité de définir une classe par énumération de ses instances et le support de types de données et de valeurs ;

D'une manière générale, les différentes extensions de SH sont généralement dénotées en juxtaposant directement la lettre calligraphique correspondante (I, F, Q, N respectivement pour le rôle inverse, fonctionnel, restriction de cardinalité qualifié ou non).

OWL DL et OWL Lite (cf. sections III.3.6.2 et III.3.6.3) peuvent être vus comme des logiques de descriptions expressives avec l'ontologie comme base de connaissance.

La spécification OWL 1.1 étend OWL en utilisant un certain nombre de constructions de la famille de logique de description SROIQ (qui ajoute principalement la réflexivité à SHOIN) qui permet de définir des types de données utilisateurs, les prédicats sur les types de données, la réflexivité, l'anti-symétrie, les restrictions de propriétés qualifiées et les restriction sur les propriétés simples

III.3.3.4 Syntaxe des Logiques de Description

Les logiques de descriptions possèdent principalement deux syntaxes : à la syntaxe ***lispienne***, où le nom des constructeurs est donné en toutes lettres et où la notation est préfixée correspond une syntaxe ***allemande***, utilisée dans la plupart des articles théoriques traitant des logiques de descriptions. Dans la suite, les deux systèmes seront utilisés indifféremment.

La figure ci-dessous (Figure 3.3) présente la grammaire du langage minimal AL exprimée en syntaxe lispienne et la syntaxe allemande correspondante.

	Syntaxe lispienne		*Syntaxe allemande*
C, D →	A		
	Top	|	$\top$
	Bottom	|	$\bot$
	(and C D)	|	$C \cap D$
	(not A)	|	$\neg A$
	(all r C)	|	$\forall r.C$
	(some r)	|	$\exists r$

Figure3.3 : Exemple de grammaire de Logique de description : le langage minimal AL

C et D sont des noms de concepts, A un nom de concept primitif et r un nom de rôle primitif.

Le constructeur *TOP* (⊤) dénote le concept le plus général et le concept *BOTTOM* (⊥) le concept le moins spécifique. Intuitivement, l'extension de *TOP* inclut tous les individus possibles tandis que celle de *BOTTOM* est vide. Le constructeur *and* (∩) permet de définir une conjonction d'expression de concepts. Le constructeur *not* (¬) correspond à la négation et ne porte que sur les concepts primitifs. La quantification universelles *all* (∀r.C) précise le co-domaine du rôle r. La quantification existentielle *some* (∃r) introduit le rôle r et affirme l'existence d'(au moins) un couple d'individus en relation par l'intermédiaire de r.

III.3.4 Identification des ressources

OWL possède plusieurs caractéristiques qui lui permettent d'être plus qu'une variante syntaxique des logiques de description. Une de ces caractéristiques est son mécanisme d'identification : pour supporter la multiplicité des ontologies sur le Web, un unique identificateur doit être associé à chaque ressource (classe, instance, propriété). Pour cela, OWL utilise les URIs (Uniform Ressource Identifiers). L'utilisation des URI permet à tout constructeur d'ontologie de référencer et d'utiliser les ressources d'autres ontologies, créant ainsi une sorte « d'ontologie globale »

Une URI peut être séparée en deux parties : le ***namespace*** et le ***nom local*** qui sont séparés par le caractère dièse. Pour des raisons de simplicité, le namespace est souvent remplacé par un nom court ou préfixe et dans ce cas, il est séparé du nom local par de le caractère « : » Par exemple, une référence au concept annuaire : http://lisi.ensma.fr/sec#annuaire sera remplacée par : « sec :annuaire ».
Comme mentionné précédemment, l'utilisation des URIs permet de créer des « ontologies globales » tout en permettant aux différentes ontologies d'être développées indépendamment et ceci à différents endroits et dans différentes organisations.

A la différence des bases de données, OWL ne fait pas l'hypothèse de l'unicité de nommage. Le fait que deux instances aient un ID différent n'implique pas qu'elles représentent des individus différents. Par exemple, si la propriété « estEnseignePar » indique qu'un cours est enseigné par au plus un enseignant, alors si l'on déclare que le cours « Bases de données » est enseigné par les enseignants 1234 et 1235, un interpréteur OWL ne conclut pas à une erreur mais à l'égalité des ressources 1234 et 1235.

III.3.5 Les constructeurs et axiomes OWL

Les constructeurs présentés dans la suite incluent ceux issus de la spécification 1.1 de OWL. Pour des raisons de lisibilité, nous utiliserons dans la suite la syntaxe des logiques de description et la syntaxe abstraite OWL.

Syntaxe DL	Linguiste = Personne ⊓ ≤ 2 parle
Syntaxe abstraite	Linguiste = intersection (Personne, Restriction (parle, minCardinality(2)))
Syntaxe RDF/XML	<owl:Class rdf:ID='Linguiste'> <owl:intersectionOf rdf:parsetype='Collection'> <owl:Class rdfs:about='Personne'> <owl:Restriction> <owl:onProperty rdf:resource='parle'/> <owl:minCardinality>2</owl:minCardinality> </owl:Restriction> </owl:intersectionOf> </owl:Class>

Tableau 3.1 : Comparaison des syntaxes de OWL

III.3.5.1 Les constructeurs de classes

OWL permet de déclarer des classes, les organiser en une hiérarchie de subsumption comme RDFS. Une classe OWL peut être spécifiée comme étant :

- Une classe nommée (en utilisant ***owl :Class*** avec une URI),
- Une expression de classe comme décrit ci-dessous.

Les classes nommées peuvent être reliées à des expressions de classes en utilisant les axiomes ***owl : subclassOf*** et ***owl :equivalentClass*** (ces axiomes sont décrits plus loin).

Syntaxe abstraite OWL	*Syntaxe DL*
A (a URI reference)	A
owl :Thing	T
owl :Nothing	$\perp$
Class(A partial C_1 … C_n)	$A \subseteq C_i$
Class(A complete C_1 … C_n)	$A \equiv C_1 \cap \ldots \cap C_n$

Tableau 3.2 : Les axiomes de classes OWL

Une expression de classe est soit :

1. $\{i_1, i_2, \ldots, i_n\}$: une énumération d'instance (en utilisant le constructeur ***owl:oneOf***). Exemple : sexe ≡ owl:oneOf(RDFSLitteral(« masculin »), RDFSLitteral(« féminin »)) ;

2. Une restriction de propriété. Il existe des restrictions de propriétés locales qui restreignent le co-domaine de certaines propriétés uniquement pour la classe où elles sont spécifiées.
   a) *P.C*, ***owl :someValuesFrom*** : au moins une des valeurs de P est une instance de la classe C.
   b) *P.C*, ***owl :allValuesFrom*** : toutes les valeurs de P sont des instances de la classe C.
   c) $i \in P$, ***owl : hasValue*** : une des valeurs de la propriété P est l'individu i.
   d) $i \notin P$, ***owl : valueNot*** : l'individu i n'est pas une valeur de P.

3. Une restriction de cardinalité qui restreint localement le nombre de valeurs distinctes d'une propriété P. On distingue :
   Les restrictions non typées qui, ne précisent pas le type du co-domaine.
   a) $\geq n$ *P*, ***owl :minCardinality*** : P possède au moins n valeurs distinctes.
   b) $\leq n$ *P*, ***owl :maxCardinality*** : P possède au plus n valeurs distinctes.
   c) $= n$ *P*, ***owl : Cardinality*** : P possède exactement n valeurs distinctes.

   Les restrictions typées qui, précisent le type du co-domaine.
   d) $\geq n$ *P.C*, ***OWL :minCardinalityQ*** : P possède au moins n valeurs distinctes de type C.
   e) $\leq n$ *P.C*, ***OWL :maxCardinalityQ*** : P possède au plus n valeurs distinctes de type C.
   f) $= n$ *P.C*, ***OWL :CardinalityQ*** : P possède exactement valeurs distinctes de type C.

4. Un constructeur booléen de classes
   a) $C_1 \cap C \cap \ldots \cap C_n$, ***owl :intersectionOf***: la sémantique de l'intersection est identique à celle de la multiple instanciation (C est l'intersection des C_i est équivalent à C est sous-classe de tous les C_j).
   b) $C_1 \cup C_2 \cup \ldots \cup C_n$, ***owl :unionOf***.
   c) $\neg C$, ***owl :ComplementOf***.

Syntaxe abstraite OWL	*Syntaxe DL*
oneOf(o_1 … o_n)	{ o_1 … o_n }
restriction(R someValueFrom(C))	∃R.C
restriction(R allValueFrom(C))	∀R.C
restriction(R value(o))	∃R.o
restriction(R minCardinality(n))	≥n R
restriction(R maxCardinality(n))	≤n R
restriction(R cardinality(n))	= n R
intersectionOf(C_1 … C_n)	$C_1 \cap \ldots \cap C_n$
unionOf(C_1 … C_n)	$C_1 \cup \ldots \cup C$
complementOf(C)	¬C

Tableau 3.3 : Les expressions de classes OWL

La hiérarchie des classes OWL est définie par les axiomes :

1. $C \subseteq D$, ***owl :subClassOf***: qui équivaut à la subsumption en logique de description. Les seuls objets qui peuvent être instances de la sous-classe C sont ceux qui sont instances de la super-classe D.
2. $C \equiv D$, ***owl :EquivalentClass*** : cet axiome équivaut à ((C ⊆ D) et (C ⊇ D)).
3. ***Owl :disjointWith*** : cet axiome équivaut à (C ∩ D) ⊆ ⊥ ;

Syntaxe abstraite OWL	*Syntaxe DL*
subClass(C_1 C_2)	$C_1 \subseteq C_2$
equivalentClasses(C_1 … C_n)	$C_1 \equiv \ldots \equiv C_n$
disjointClasses(C_1 … C_n)	$C_1 \cap C_2 \subseteq \perp$

Tableau 3.4 : Les axiomes de hiérarchie de classe OWL

OWL distingue une classe primitive d'une classe définie comme suit :

- Une classe primitive spécifie une condition nécessaire pour tester l'appartenance d'un individu. Une classe primitive est déclarée par les axiomes owl :class et/ou rdfs :subClass uniquement .

- Une classe définie spécifie une condition nécessaire et suffisante pour tester l'appartenance d'un individu. Les individus d'une telle classe sont donc inférés. Une classe définie est déclarée soit par les axiomes owl :oneOf, owl : equivalentClass owl :complementOf ou par les axiomes owl : unionOf et owl : intersectionOf .

III.3.5.2 Les constructeurs de propriétés

OWL permet aussi de déclarer des propriétés et de les organiser en hiérarchie de « sous-propriétés ».
Une propriété OWL peut être spécifiée comme étant :

- Une propriété simple (en utilisant le constructeur ***owl : datatypeProperty***) : son co-domaine est alors un type de données dit simple et issu de la spécification XML Schema.
- Une propriété complexe (en utilisant le constructeur ***owl : ObjectProperty***) : son co-domaine est une classe de l'ontologie.

Contrairement à l'approche utilisée par les langages de programmation orientés objets et dans les langages d'ontologies de type « frame-based », les propriétés dans les logiques de description ne sont pas définies comme des parties des classes. De ce fait, plusieurs classes peuvent partager la même propriété. L'association des propriétés aux classes se fait lors de la définition du domaine de la propriété (***rdfs :domain***). De la même manière, une propriété peut préciser explicitement son co-domaine (***rdfs :range***).

Notons que le domaine et le co-domaine d'une propriété OWL sont interprétés comme une intersection.

Comme pour les classes, OWL définit des axiomes pour hiérarchiser les propriétés :

1. $\subseteq$, ***owl :subPropertyOf*** : qui équivaut à la subsumption de propriétés en logique de description.
2. $\equiv$, ***owl :Equivalentproperties***. : qui spécifie que des propriétés ont la même extension.

En plus, OWL supporte différentes caractéristiques (mathématiques) associées aux propriétés :

1. ***owl :ReflexiveProperty*** : définit une propriété transitive, comme « estAmiDe ».
2. ***owl :TransitiveProperty*** : définit une propriété transitive, comme « estPlusGrandQue ».
3. ***owl :SymmetricProperty*** : définit une propriété symétrique, comme «aLeMemeGradeQue».
4. ***owl irReflexiveProperty*** : définit une propriété irréflexive, comme «aleMemeGradeQue»
5. ***owl :antiSymmetricProperty*** : définit une propriété anti-symétrique, comme «estPlusGrandQue».
6. ***owl :InverseOf*** : définit une propriété comme étant l'inverse d'une autre propriété : « aPourEnfant » et «aPourParent».
7. ***owl :FunctionalProperty*** : la propriété ne pas avoir plus d'une valeur comme «estNéLe».
8. ***owl :InverseFunctional*** : une unique ressource peut être associée à la valeur de cette propriété : «aPourNoSecuriteSociale»

Syntaxe abstraite OWL	*Syntaxe DL*
datatypeProperty(U	
super(U_1)…super(U_n)	$U \subseteq U_i$
domain(C_1)…domain(C_n)	$T \subseteq \forall U^-.C_i$
range(D_1)…range(D_n)	$T \subseteq \forall U.D_i$
[functional])	$T \subseteq 1\ U^-$
objectProperty(R	
Super(R_1)…super(R_n)	$R \subseteq R_i$
domain(C_1)…domain(C_n)	$T \subseteq \forall R^-.C_i$
range(C_1)…range(C_n)	$T \subseteq \forall R.C_i$
[transitive]	Trans(R)
[Symmetric]	$R \equiv R^-$
[inverseOf(R_0)]	$R \subseteq R^-_0$
[functional]	$T \subseteq 1\ R^-$
[inverseFunctional])	$T \subseteq 1R$
subProperty(Q_1 Q_n)	$Q_1 \subseteq Q_n$
equivalentProperties(Q_1 … Q_n)	$Q_1 \equiv \ldots \equiv Q_n$
disjointProperties(Q_1 … Q_n)	$Q_i \cap Q_j \subseteq \perp$

Tableau 3.5 : les axiomes de propriété OWL

Dans ce tableau , C fait référence à une description, D fait référence à un type de données, R fait référence à une propriété objet, U fait référence à une propriété simple et, Q fait référence à une propriété objet ou simple.

III.3.5.3 Les types de données

Les types de données supportés par OWL sont :

1. Les types de données primitifs issus de la spécification XMLSchema. On peut utiliser les types de données suivants du schéma XML comme étant des types de données intégrés OWL avec un appel d'adresse URI canonique du type de donnée du schéma XML, à savoir http://www.w3.org/2001/XMLSchema#nom, où nom représente le nom local du type de donnée : xsd:string, xsd:boolean, xsd:decimal, xsd:float, xsd:double, xsd:dateTime, xsd:time, xsd:date, xsd:gYearMonth, xsd:gYear, xsd:gMonthDay, xsd:gDay, xsd:gMonth, xsd:hexBinary, xsd:base64Binary, xsd:anyURI, xsd:normalizedString, xsd:token, xsd:language, xsd:NMTOKEN, xsd:Name, xsd:NCName, xsd:integer, xsd:nonPositiveInteger, xsd:negativeInteger, xsd:long, xsd:int, xsd:short, xsd:byte, xsd:nonNegativeInteger, xsd:unsignedLong, xsd:unsignedInt, xsd:unsignedShort, xsd:unsignedByte et xsd:positiveInteger. Les autres types de données intégrés du schéma XML sont problématiques pour le langage OWL [RDF 03]. Le type de donnée RDF intégré rdf:XMLLiteral est également un type de donnée OWL intégré. À cause de l'absence d'un moyen standard pour aller d'un appel d'adresse URI à un type de donnée du schéma XML dans XML Schema, il n'y a pas non plus de moyen standard pour utiliser les types de données du schéma XML définis par l'utilisateur dans OWL.
2. Les énumérations (construites en utilisant ***owl :oneOf***)
3. Les types de données personnalisés obtenus par application de certaines facettes. Les facettes autorisées sont : length, minLength, maxLength, pattern, enumeration, maxInclusive, maxExclusive, minInclusive, minExclusive, totalDigits, fractionDigits. Ces facettes ont la même signification que dans XML Schema et doivent être utilisées conformément aux recommandations de XML Schema à l'exception des facettes length, minLength, maxLength, et pattern qui ne sont pas permisses sur les types de données numériques. Si une facette est utilisée de manière non conforme (par exemple (length "5"^^xsd:string) ou (datatype xsd:string (maxInclusive "5"^^xsd:int))) alors l'extension du type de données correspondant est vide. rdfs:Literal ne possède pas de facettes.

III.3.5.4 Les individus

Le constructeur d'individus OWL est : *owl : individual.*

Les instances d'une classe sont définies en explicitant la ou les classes auxquelles elles appartiennent et en indiquant les valeurs de propriétés sous la forme d'une liste de couples (propriété, valeur).

Dans les logiques de description, la partie de l'ontologie qui contient les descriptions des classes est appelée la TBOX ou terminology box, en plus de cela, les assertions sur individus sont stockées dans l'ABOX ou assertion box.

Une ABOX OWL contient les assertions suivantes :

1. $i \in C$, ***rdf :type*** : permet de déclarer que l'individu i appartient à la classe C
2. $<i_1, i_2> \in P$, $<i, v> \in P$: définie la valeur d'une propriété de type complexe et simple respectivement.

3. $i_1 = i_2$, ***owl :sameAs*** : égalité entre individus
4. $i_1 \neq i_2$, ***owl :differentFrom*** : différence entre individus

Syntaxe abstraite OWL	*Syntaxe DL*
individual(o	
type(C_1)…type(C_n)	$o \in C_i$
value(R_1 o_1)… value(R_m o_m)	$\langle o, o_i \rangle \in R_i$
value(U_1 t_1)… value(U_m t_m)	$\langle o, t_i \rangle \in U_i$
sameIndividual(o_1… o_n)	$o_1 = \ldots = o_n$
differentIndividuals(o_1… o_n)	$o_1 \neq o_n$ $i \neq j$

Tableau 3.6 : assertions sur les individus OWL

Dans ce tableau , C fait référence à une description, o fait référence à un individu , t fait référence à une valeur concrète, R fait référence à une propriété objet et, U fait référence à une propriété simple.

Les valeurs de types d'individus : *10^^xsd :integer* par exemple précise que le type du littéral « 10 » est décrit par l'URI « xsd :integer ».

III.3.5.5 Autres concepts

OWL permet de construire une ontologie comme étant un ensemble de définitions de classes et de propriétés. Une ontologie peut importer (owl :imports) l'ensemble des concepts définis par une autre ontologie. Une ontologie, tout comme les classes, propriétés et instances, peut être annotée. Ceci permet d'associer à un tel concept un mot (label), un numéro de version (versionInfo), un commentaire (comment), des références vers d'autres concepts (seeAlso) ou même son créateur (isDefinedBy).
Les possibilités de gestion des versions d'une ontologie sont étoffées par les constructeurs :

- priorVersion : définition d'une ontologie comme étant une version précédente de celle-ci;
- backwardCompatibleWith : l'ontologie spécifiée est une version précédente de celle-ci et peut être remplacée par celle-ci (c'est-à-dire qu'elle est compatible avec sa version précédente);
- incompatibleWith : l'ontologie spécifiée est une nouvelle version de celle-ci mais des incompatibilités peuvent exister entre ces deux ontologies.

Pour le versionnement, les deux constructeurs DeprecatedProperty et DeprecatedClass permettent d'indiquer que la propriété ou la classe ne doit plus être utilisée ou que leur usage est désapprouvé.

III.3.6 Les sous-langages d'OWL

OWL établit un compromis entre son pouvoir expressif et son pouvoir de raisonnement en fournissant trois sous-langages de compatibilité et d'expressivité croissante : OWL Lite, OWL DL et OWL Full.

III.3.6.1 OWL Full

Il inclut toutes les primitives OWL qui peuvent être combinées avec toutes les primitives RDF et RDFS. Il a l'avantage de la compatibilité complète avec RDF/RDFS, mais l'inconvénient d'avoir un niveau d'expressivité qui le rend indécidable.
Le langage OWL Full permet aux classes d'apparaître comme des individus, aussi, les propriétés objets et les propriétés simples ne sont pas disjointes.

Le langage OWL Full se destine typiquement aux personnes qui veulent combiner l'expressivité du langage OWL à la souplesse et aux capacités de méta-modélisation de RDF. Cependant, utiliser les fonctionnalités de OWL Full signifie aussi perdre quelques unes des garanties (voir ci-dessous) que OWL DL et OWL Lite peuvent offrir aux systèmes de raisonnement.

III.3.6.2 OWL DL

Basé sur la logique de description SHOIN, il définit des restrictions sur la manière dont les primitives OWL et RDF/RDFS peuvent être combinées. Ce niveau de langage est décidable. Il a l'inconvénient de ne pas être complètement compatible avec RDF. Un document RDF devra donc parfois être étendu sur certains aspects ou restreint sur d'autres pour être un document OWL-DL légal.

- OWL DL demande une séparation deux à deux entre les classes, les types de données, les propriétés de type de données, les propriétés d'objet, les propriétés d'annotation, les propriétés d'ontologie (c'est-à-dire, ce qui concerne l'importation et le versionnage), les individus, les valeurs de données et le vocabulaire intégré. Cela signifie que, par exemple, une classe ne peut pas être dans le même temps un individu.

- Dans OWL DL, l'ensemble des propriétés objets et celui des propriétés simples sont disjoints pour les quatre caractéristiques suivantes : réflexive, symétrique, inverse fonctionnelle et transitive.

- OWL DL interdit d'exercer une contrainte de cardinalité (ni locale ni globale) sur les propriétés transitives ou leurs symétriques ou leurs éventuelles sous-propriétés.

- Tous les axiomes doivent être bien formés, pas de composant manquant ou en trop, et doivent former une structure arborescente.

- Cette dernière contrainte implique que toutes les classes et propriétés appelées soient explicitement typées comme étant, respectivement, des classes ou des propriétés OWL. Par exemple, si l'ontologie contient le composant suivant :
  ```
  <owl:Class rdf:ID="C1">
        <rdfs:subClassOf rdf:resource="#C2" />
  </owl:Class>
  ```
  Alors l'ontologie (ou une ontologie importée dans celle-ci) devrait contenir un triplet `owl:Class` pour `C2`.

Les restrictions OWL DL permettent de constituer le sous-ensemble maximal du langage OWL Full, par rapport auquel les travaux actuels peuvent garantir l'existence d'une procédure de raisonnement décidable pour un raisonneur OWL.

III.3.6.3 OWL lite

Basé sur la logique de description SHIF, OWL Lite est une version limitée d'OWL-DL qui en facilite son utilisation et son implémentation.
Le langage OWL Lite interdit, en outre, l'emploi de : owl:oneOf, owl:unionOf, owl:DataRange, owl:complementOf, owl:hasValue, owl:disjointWith.

Le langage OWL Lite exige aussi que :

- les sujets des triplets owl:equivalentClass soient des noms de classe et leurs objets des noms de classe ou des restrictions ;
- les sujets des triplets rdfs:subClassOf soient des noms de classe et leurs objets des noms de classes ou des restrictions ;
- les relations owl:intersectionOf soient seulement utilisées sur des listes dont la longueur est supérieure à un et qui ne contiennent que des noms de classe ;
- les objets des triplets owl:allValuesFrom et owl:someValuesFrom soient des noms de classe ou des noms de type de données ;
- les objets des triplets rdf:type soient des noms de classe ou des restrictions ;
- les objets des triplets rdfs:domain soient des noms de classe, et
- les objets des triplets rdfs:range soient des noms de classe ou des noms de type de donnée.

L'idée derrière les limitations de l'expressivité du langage OWL Lite est celle selon laquelle ces limitations produisent un sous-ensemble pratique minimal des caractéristiques du langage dont la mise en œuvre est simple pour les développeurs d'outils.

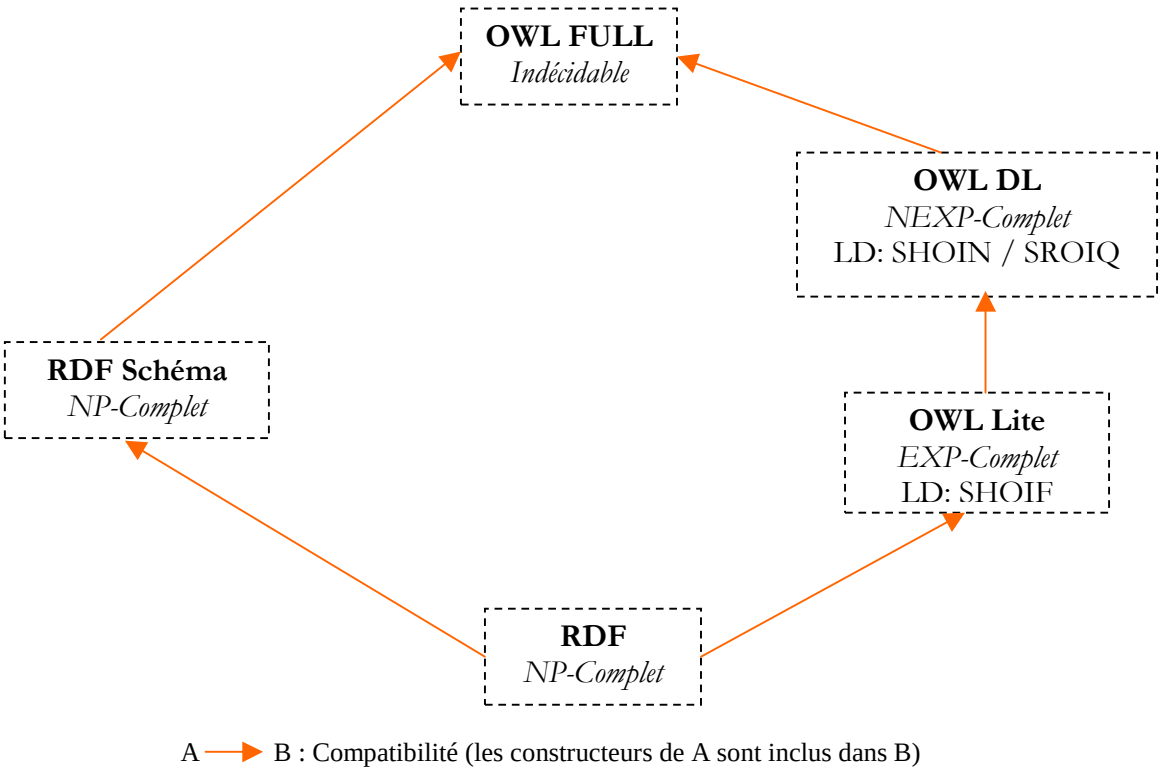


Figure 3.4 : Les sous-langages de OWL

III.3.7 Extensions prévues d'OWL [Pan & Horrocks 04]

Récemment, Pan et Horrocks on proposée une extension de OWL qui a pour but de résoudre les problème ci-dessous cités :

- Contrairement aux classes et propriétés, le pouvoir d'expression des types de données de OWL est faible. Cela est particulièrement évident quand on veut faire la distinction entre des données sémantiquement différentes, même si elles sont syntaxiquement équivalentes. Par exemple, une chaîne représentant un nom doit être sémantiquement différente d'une chaîne représentant un matricule ou un code même si elles sont toutes deux des chaînes.

- Les types de données devraient être organisés en une hiérarchie extensible afin que les types personnalisés puissent être considérés comme des sous-types des types de base ou même d'autres types personnalisés. Par exemple, un nouveau type de données « mesure » peut être déclaré comme dérivé d'Entier, et d'autres nouveaux types « mesureCm » et « mesureInch » comme ses sous types.

- Il devrait être possible de créer de nouveaux types de données en appliquant divers types de contraintes aux types existants.

- Il devrait être possible de créer des types de données par combinaison de divers autres types préexistant en utilisant des opérateurs booléens.

- Il serait utile de permettre l'importation de contraintes sémantiques sur les types de données, comme c'est le cas pour les classes. Par exemple on aimerait être capable de décrire sémantiquement un service qui prend le coût HT d'un composant en EUROS et retourne le coût TTC en DOLLARS ou encore, le rayon d'un objet circulaire en cm carrées et retourne son volume en square Inches.

- On aimerait également établir des équivalences équationnelles entre des propriétés : (diamètre = rayon * 2).

Le principal problème réside dans le fait que dans OWL, les types de données sont traités différemment que les classes, ils ne peuvent pas être définis en utilisant les expressions, ne peuvent avoir des propriétés et ne peuvent être « subClassOf », « equivalentClass », ou « disjointClass » avec d'autres types de données.

Dans leur proposition appelée OWL-E , Pan et Horrocks introduisent une méthode générale pour représenter les types de données personnalisés. Ceci en enrichissant OWL avec la notion de « **prédicat de type de données** ou **datatype predicate**» tels que (<, =) et de la notion d' « **expression de types de données ou datatype expression** ». La combinaison de ces 2 notions permet la création de types de données personnalisés incluant les types complexes nécessitant des combinaisons de prédicats booléens (ie : x<10 & x>=100).

Cela est fait en gardant la distinction entre type de données et classes mentionnée précédemment en utilisant la notion de « groupe de type de données » comme sémantique sous-jacente. Afin de permettre l'utilisation de types de données personnalisées dans la construction des classes (pour construire une classe comme Adult ≡ Person ∩ ∀ age(<18)) , la proposition de Pan et Horrocks

introduisent quatres nouvelles restrictions qui correspondent à ∀ , ∃, <=, >=. Enfin, ils démontrent que OWL-E est décidable ; ie : des raisonneurs pour inférer sur OWL-E peuvent être implémentés.

OWL-E n'est pas encore un standard et ne possède actuellement pas de supports d'APIs comme Jena, ni de raisonneur associé. Enfin, le point 4) n'est pas résolu par OWL-E qui conserve toujours la distinction entre classes et types de données.

Conclusion

OWL et RDF Schema sont tous deux des vocabulaires RDF permettant de définir d'autres vocabulaires. RDF Schema définit le plus petit nombre de notions et de propriétés nécessaires à la définition d'un vocabulaire simple, essentiellement :

- les notions de classe, ressource, littéral;
- les propriétés de sous-classe, de sous-propriété, de champ de valeur, de domaine d'application.

OWL est un langage beaucoup plus riche qui, aux notions définies par RDF Schema, ajoute les propriétés de classe équivalente, de propriété équivalente, d'identité de deux ressources, de différences de deux ressources, de contraire, de symétrie, de transitivité, de cardinalité, etc., permettant de définir des rapports complexes entre des ressources**.**

L'intention de OWL est identique à celle de RDF-S qui est de fournir un vocabulaire XML pour définir les classes, propriétés et leurs relations.

- RDF-S permet d'exprimer des relations rudimentaires sans inférence.
- OWL permet d'exprimer des relations plus riches et engendrer ainsi des inférences.

L'avantage le plus important de OWL est de créer des inférences d'un niveau plus élevé que RDF-Schema. OWL est une "Extension" de RDFS pour faciliter :

1. le partage/l'intégration d'ontologies et de méta-données.
2. l'évolution d'ontologies.
3. le raisonnement : détection d'inconsistances, …

Pour la comparaison entre PLIB et OWL, nous nous limiterons aux versions Lite et DL de ce dernier qui permettent la distinction entre les notions d'instances et de classes. Dans la suite de ce chapitre, sauf indication contraire, nous présenterons la version DL de OWL.

CHAPITRE IV : CONVERSION DE SCHEMA ENTRE MODELES DE DONNEES

Rappelons que le but de notre travail est d'élaborer une démarche pour transformer et échanger les informations entre ontologies issues des approches de modélisation basés sur la logique de description (ontologies construites à partir du langage OWL) et les ontologies issues des approches modélisation basées sur les bases de données (ontologies construites à partir du modèle OWL). Il s'agit donc de concevoir des règles permettant la transformation systématique d'un dictionnaire PLIB en une ontologie OWL (et inversement, la transformation d'une ontologie OWL en un dictionnaire PLIB).

Deux approches sont généralement utilisées pour transposer les données:

1. une vue : qui permet :
    - d'extraire d'un modèle de données ce qui est pertinent pour chaque catégorie d'utilisateur ;
    - d'autoriser l'accès en fonction de l'utilisateur ;
    - de ne pas modifier les programmes lors de l'évolution du schéma (re-compilation de la vue à chaque évolution).

    En utilisant les vues ou schémas externes les données ne sont pas physiquement converties

2. une correspondance ou « mapping » : qui permet:
    - d'intégrer des données correspondant au même modèle ;
    - d'utiliser des modèles internes différents des modèles d'échanges (modèle source et modèle cible).

    En utilisant un mapping ou schéma interne, les données sont physiquement converties

D'une manière générale les vues diffèrent des mappings en ce que les vues ne mettent pas « en jeu » des schémas cibles, ni un ensemble de données cibles. C'est la déclaration d'une vue qui définie la structure de l'information qui est la cible de la transformation. Les mapping quant à eux consistent à transformer les instances de concepts d'un modèle X dit modèle source vers les instances de concepts d'un autre modèle Y dit modèle cible, X et Y étant définis dans le même formalisme ou instances d'un même méta-modèle.

Le formalisme de modélisation utilisé dans PLIB est le langage EXPRESS qui possède à la fois une version textuelle et une version graphique, similaire à UML. OWL utilise quant à lui soit le formalisme des logiques de description, soit la syntaxe abstraite RDF/XML ou une syntaxe abstraite. Dans tous les cas, la description d' OWL est textuelle.

Vu que pour spécifier des règles de transformation permettant de représenter physiquement des ontologies PLIB en des ontologies OWL (et inversement), il est nécessaire d'exprimer ces deux modèles d'ontologies en utilisant le même formalisme, nous avons été amené à choisir un formalisme de modélisation commun pour ces deux modèles.

IV.1 Choix du formalisme de modélisation

Nous avons choisi le langage EXPRESS qui est un langage ensembliste et orienté objet de spécification de données associé à des mappings (génération automatique de modèles de données). C'est aussi un ensemble complet intégrant à la fois, un langage de modélisation orienté objet, un outil de vérification de modèle et de spécification des contraintes, et, version graphique EXPRESS-G qui permet de donner une vue synthétique et facilement lisible des modèles de données.

EXPRESS [Hondjack 05] est également une série de normes définissant un environnement constitué de :

- un langage de modélisation de l'information : EXPRESS (ISO 10303-11 :1994) ;
- un format d'instances qui permet l'échange de données entre systèmes (ISO 10303-21 :1994) ;
- une infrastructure de méta-modélisation associée à une interface d'accès normalisée, appelée SDAI, pour accéder et manipuler simultanément les données et le modèle de n'importe quel modèle EXPRESS. Cette interface associé à un méta-modèle d'EXPRESS en EXPRESS a d'abord été définie indépendamment de tout langage de programmation (ISO 10303-22 :1998) puis des implémentations spécifiques ont été spécifiées pour le langage C++ (2000), Java (2000), C (2001) ;
- un langage déclaratif de transformation de modèles (ISO 10303-14 :2002).

Enfin, le langage EXPRESS possède un langage procédural complet (analogue à PASCAL) pour l'expression de contraintes. Au prix de quelques extensions mineures, ce langage peut également être utilisé comme langage impératif de transformation de modèles.

Depuis une dizaine d'années, de nombreux environnements de modélisation EXPRESS sont commercialisés et le langage EXPRESS est utilisé dans de nombreux domaines, que se soit pour modéliser et échanger des descriptions de produits industriels, ou pour spécifier des bases de données dans des domaines divers, ou même pour fabriquer des générateurs de codes dans des ateliers de génie logiciel.

PLIB étant modélisé en utilisant le formalisme EXPRESS, nous devons donc concevoir un modèle EXPRESS pour OWL.

Ainsi :

- La transformation d'une ontologie PLIB vers une ontologie OWL revient à établir un mapping entre le modèle PLIB et le schéma EXPRESS d'OWL. Et,

- La transformation d'une ontologie OWL vers une ontologie PLIB revient uniquement établir un mapping entre le modèle EXPRESS d'OWL et le modèle PLIB car ce dernier est modélisé en utilisant le formalisme EXPRESS.

Avant de proposer un modèle EXPRESS pour OWL, nous commençons d'abord par une présentation de formalisme de modélisation EXPRESS (plus précisément) de sa version graphique que nous allons utiliser dans la suite pour illustrer nos exemples).

IV.2 Le langage EXPRESS

IV.2.1 Introduction [PIE 02]

EXPRESS est un langage de modélisation de données conçu dans le dans le cadre du projet STEP [PIE00]. L'objectif d'EXPRESS [ISO10303-11: 1994] est la description de modèles d'informations en vue de l'échange de données représentant de façon fiable et non ambiguë ces informations.

Dans le langage EXPRESS, l'accent principal est mis sur la précision du modèle et tout particulièrement sur les contraintes que doivent respecter les données pour être acceptées comme conformes au modèle. Ceci assure la fiabilité de l'information représentée.

Un modèle EXPRESS, également appelé schéma, définit un ensemble d'entités qui représentent les objets à modéliser associé à un mécanisme de généralisation / spécialisation. Chaque entité est définie par un ensemble d'attributs. Chaque attribut possède un domaine de valeurs pouvant être un type simple (entier, réel, énumération,..), un type entité, une union de types ou un agrégat (ensemble, liste, ...). Des contraintes fonctionnelles et assertionnelles restreignent les interprétations du modèle.

EXPRESS possède à la fois une version textuelle une version graphique EXPRESS-G, similaire à UML.

Dans ce langage, l'accent est mis sur la précision du modèle et tout particulièrement sur les contraintes que doivent respecter les données pour être acceptées comme conforme au modèle. Ceci assure la fiabilité de l'information représentée. Express n'est pas seulement une notation permettent la modélisation des données , c'est à dire une représentation simplifiée, éventuellement ambiguë, des informations propres à un domaine à fin d'échange entre concepteurs humains pour décider des éléments pertinents et des détails qu'il convient de négliger. C'est aussi un formalisme de spécification, c'est-à-dire qu'il permet une description complètement non ambiguë et traitable en machine.

Dans le cadre de notre étude, nous avons préférée la notation graphique d'EXPRESS car elle est facilement lisible contrairement à la représentation textuelle.

IV.2.1 La représentation graphique EXPRESS-G

EXPRESS-G permet de donner une vue synthétique des modèles de données et faciliter leur conception dans les phases initiales d'analyse des problèmes à modéliser. Il faut toutefois noter que ce formalisme de modélisation est partiel et ne permet pas d'exprimer les contraintes d'intégrité. La nature des attributs (optionnel, inverse (réciproque d'un autre attribut) ou dérivée (dont la valeur est une fonction des valeurs d'autres attributs), ainsi que leur type peuvent par contre être représentés.

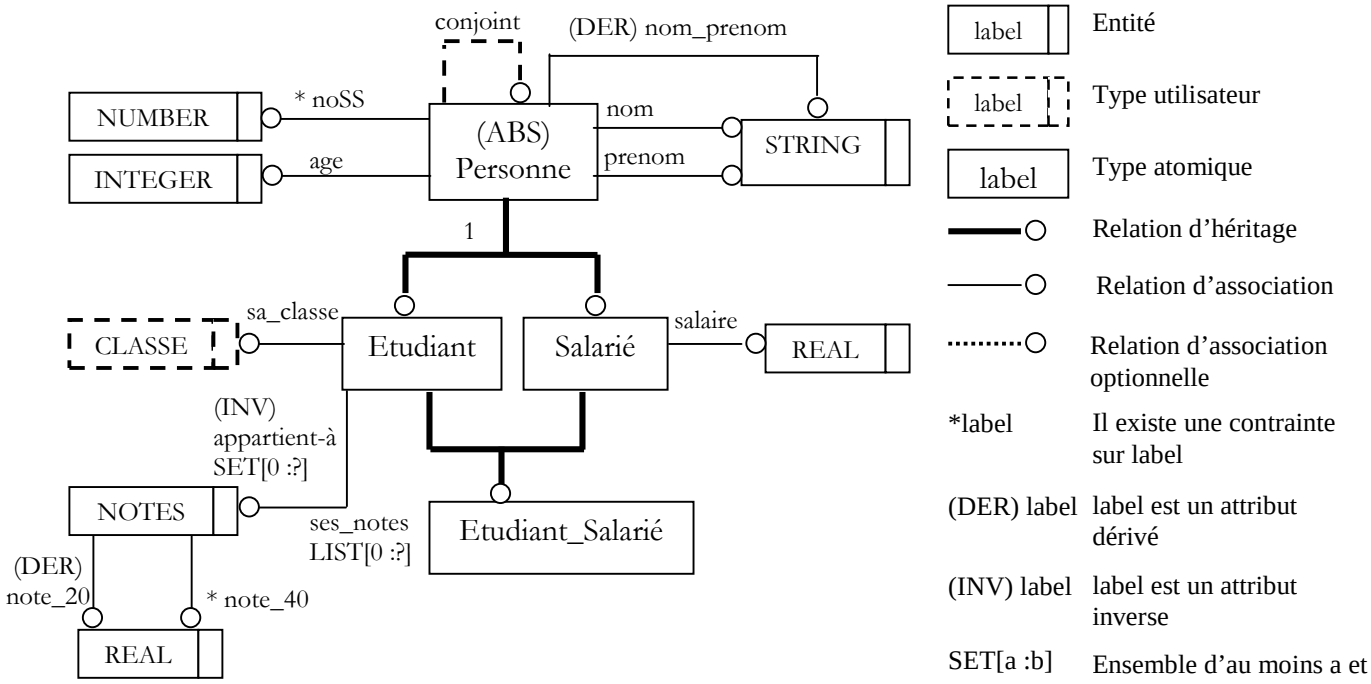


Figure 4.1 : Exemple de schéma EXPRESS-G

IV.3 Le schéma EXPRESS-G d'OWL

Ce modèle a été construit à partir des spécifications présentées dans le chapitre III.

OWL, utilise le même vocabulaire que celui de RDFS Schéma.

IV.3.1 Schéma EXPRESS-G de RDFS

La figure ci-dessous présente le modèle EXPRESS de RDF Schéma

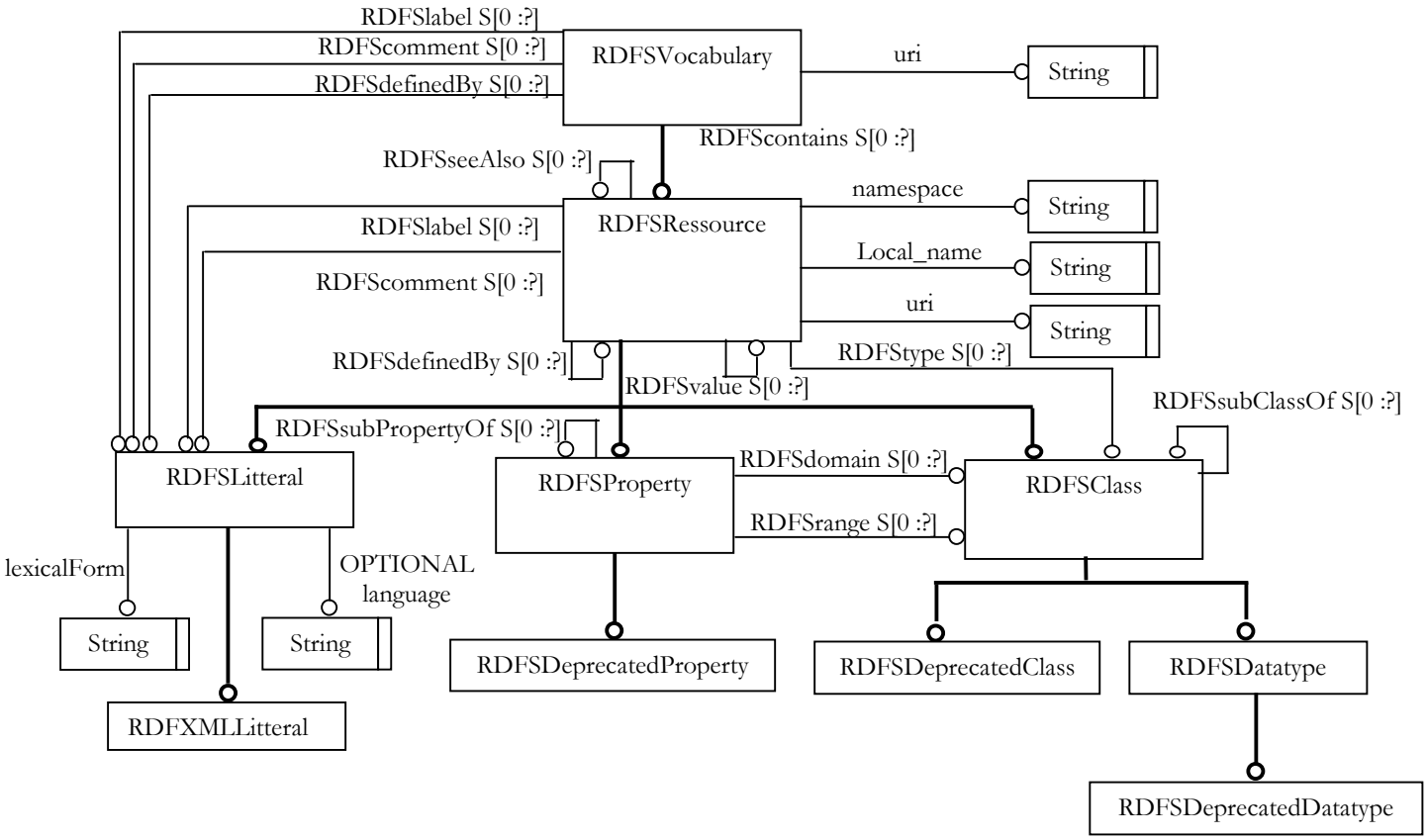


Figure 4.2 : Schéma EXPRESS-G de RDFS

IV.3.2 Schéma EXPRESS-G d'OWL : ontologie

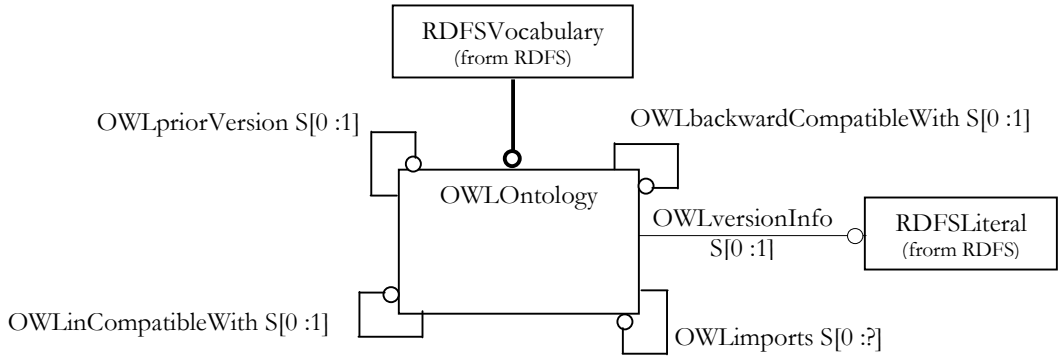


Figure 4.3 : Schéma EXPRESS-G d'OWL : ontologie.

IV.3.3 Schéma EXPRESS-G d'OWL : classe

Dans la figure ci-dessous, l'entité OWLThing représente l'ensemble des instances.

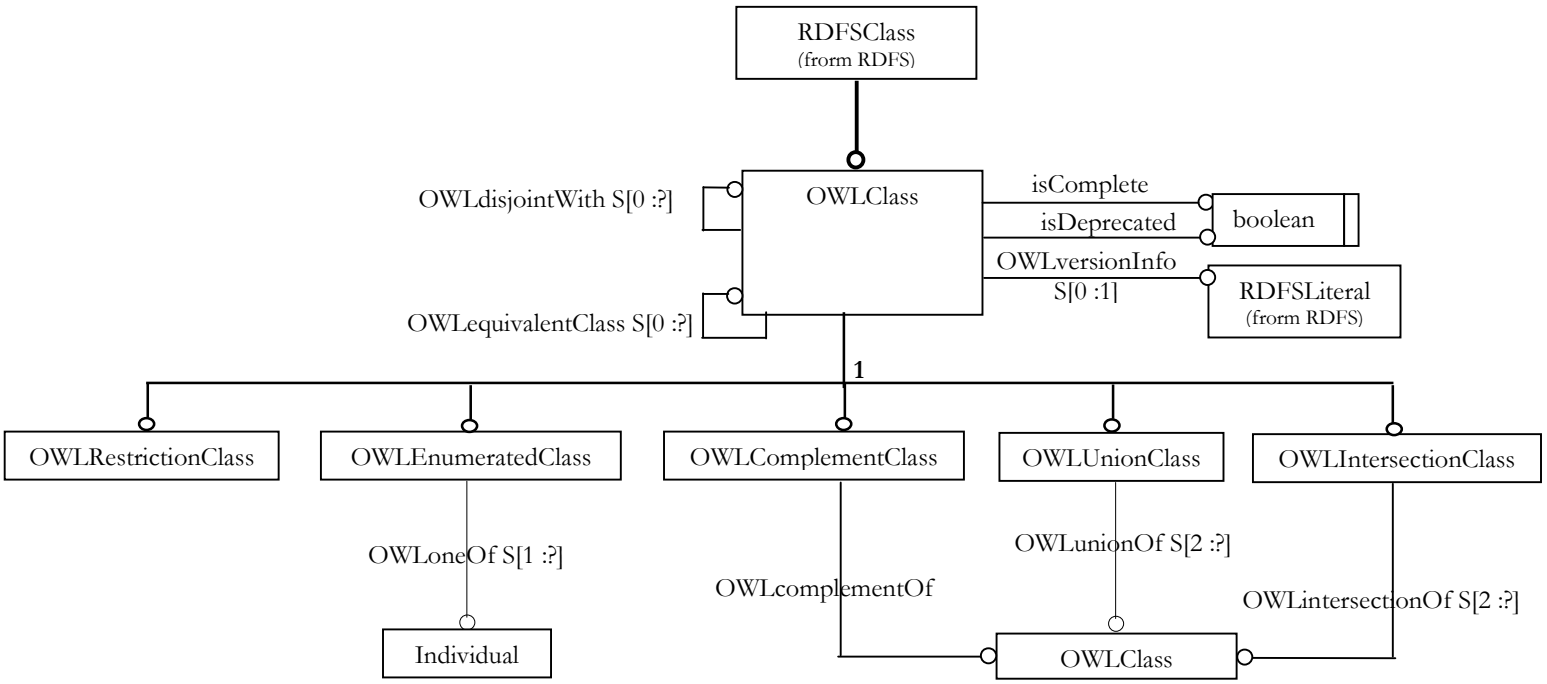


Figure 4.4 : Schéma EXPRESS-G d'OWL : classe

IV.3.4 Schéma EXPRESS-G d'OWL : restriction

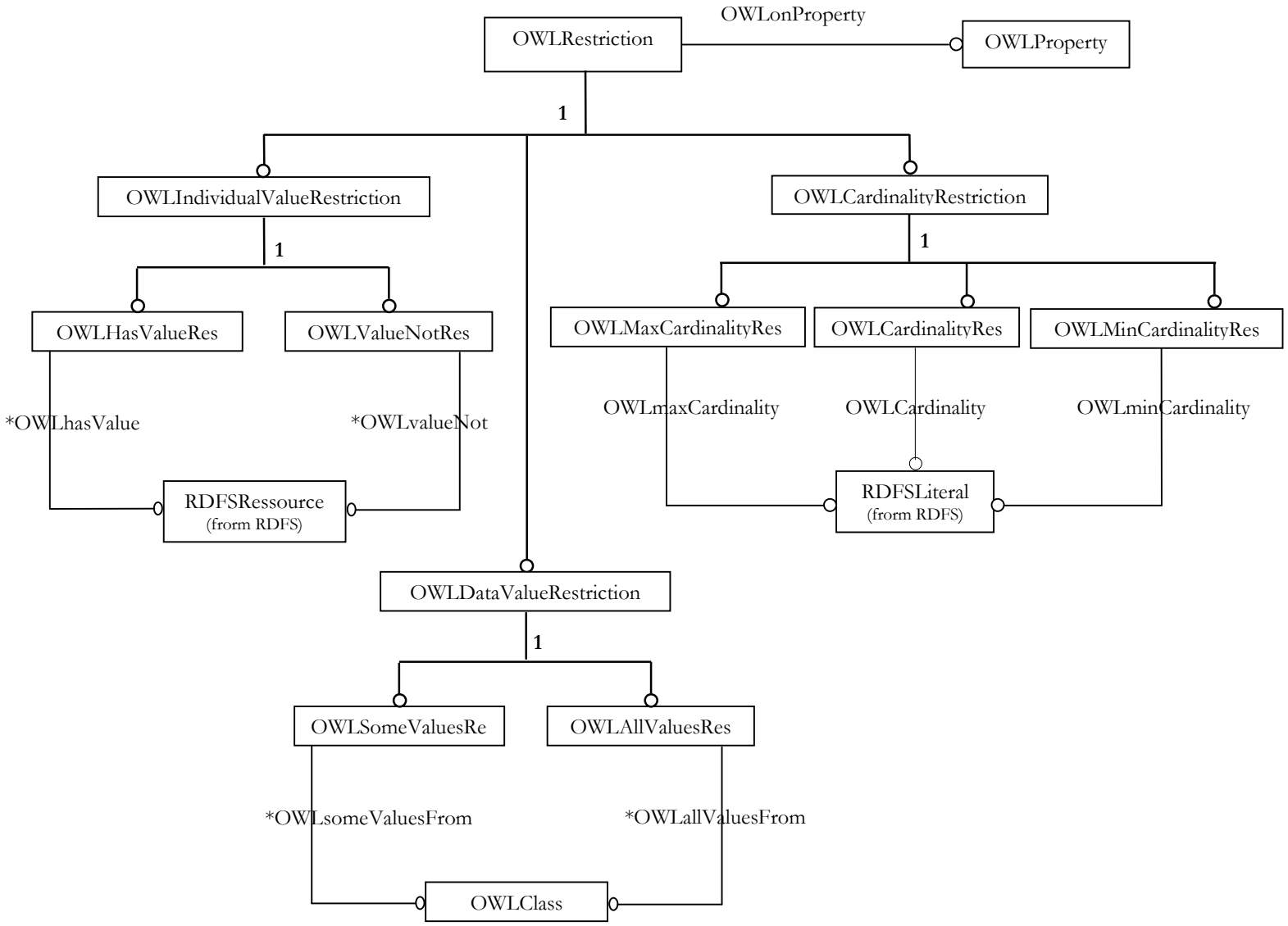


Figure 4.5 : Schéma EXPRESS-G d'OWL : restriction

Si (OWLonProperty est de type OWLObjectProperty) Alors
OWLsomeValueFrom doit être de type OWLClass
OWLallValueFrom doit être de type OWLClass
OWLhasValue doit être de type Individual
OWLvalueNot doit être de type Individual

Sinon (OWLonProperty est de type OWLDataProperty)
OWLsomeValueFrom doit être de type OWLDatarange
OWLallValueFrom doit être de type OWLDatarange
OWLhasValue doit être de type RDFSLiteral
OWLvalueNot doit être de type RDFSLiteral

IV.3.5 Schéma EXPRESS-G d'OWL : propriété

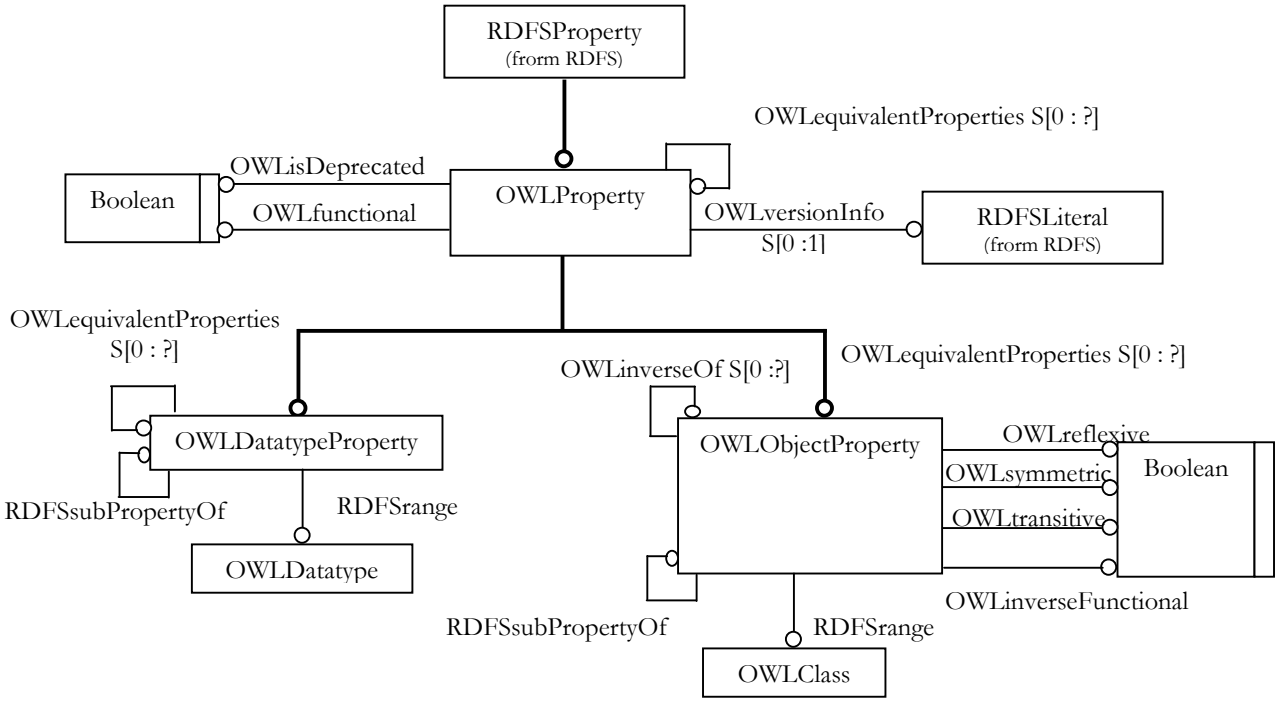


Figure 4.6 : Schéma EXPRESS-G d'OWL : propriété

IV.3.6 Schéma EXPRESS-G d'OWL : instance

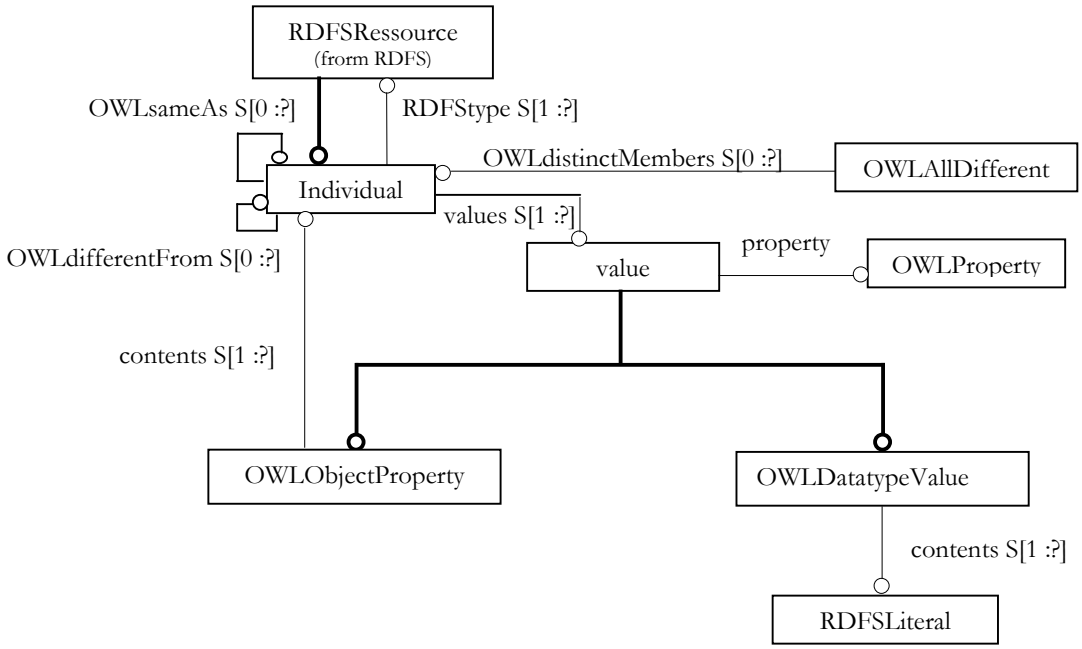


Figure 4.7 : Schéma EXPRESS-G d'OWL : instance

IV.3.7 Schéma EXPRESS-G d'OWL : types de données (1)

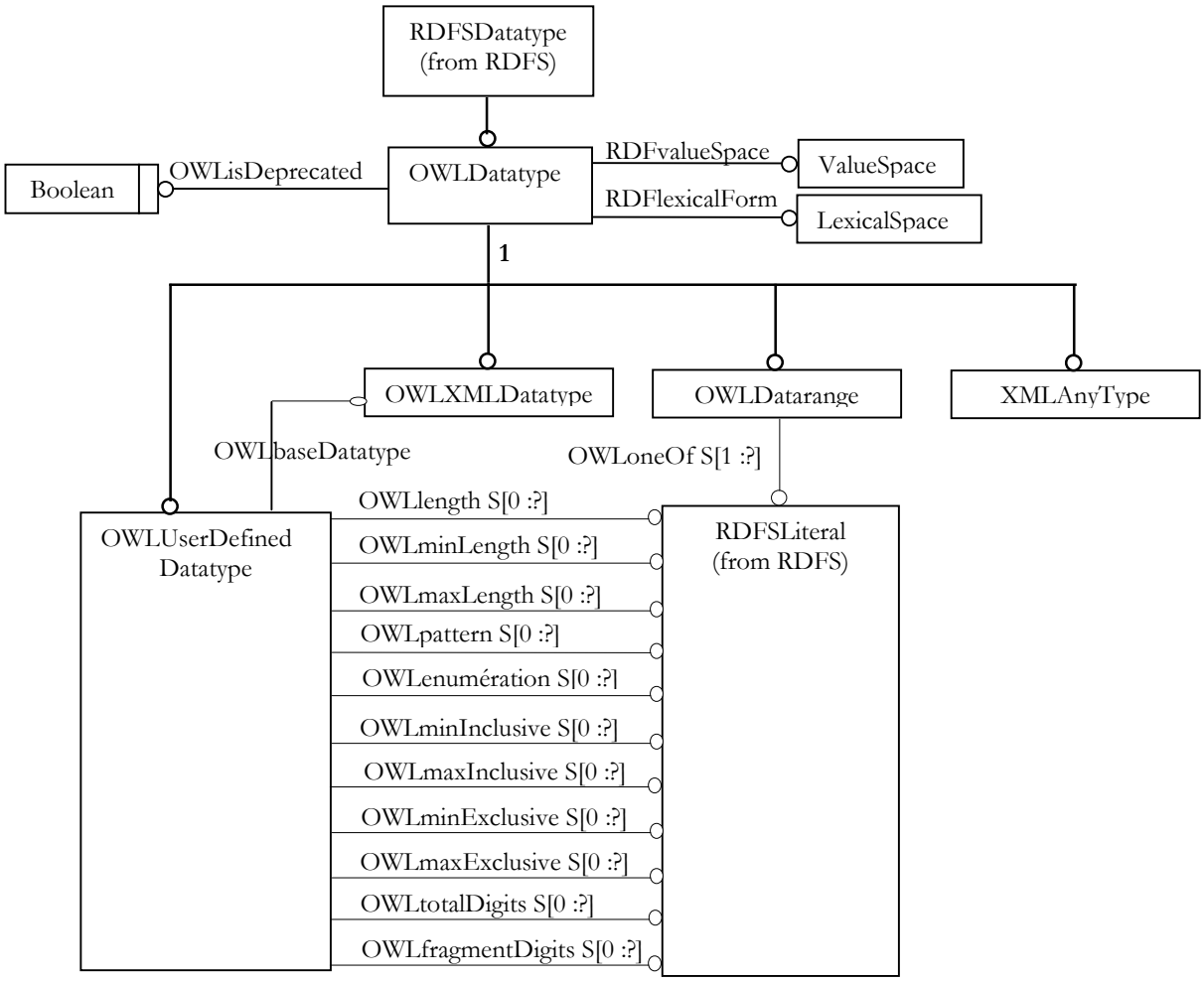


Figure 4.8 : Schéma EXPRESS-G d'OWL : types de données

IV.3.8 Schéma EXPRESS-G d'OWL : types de données (2)

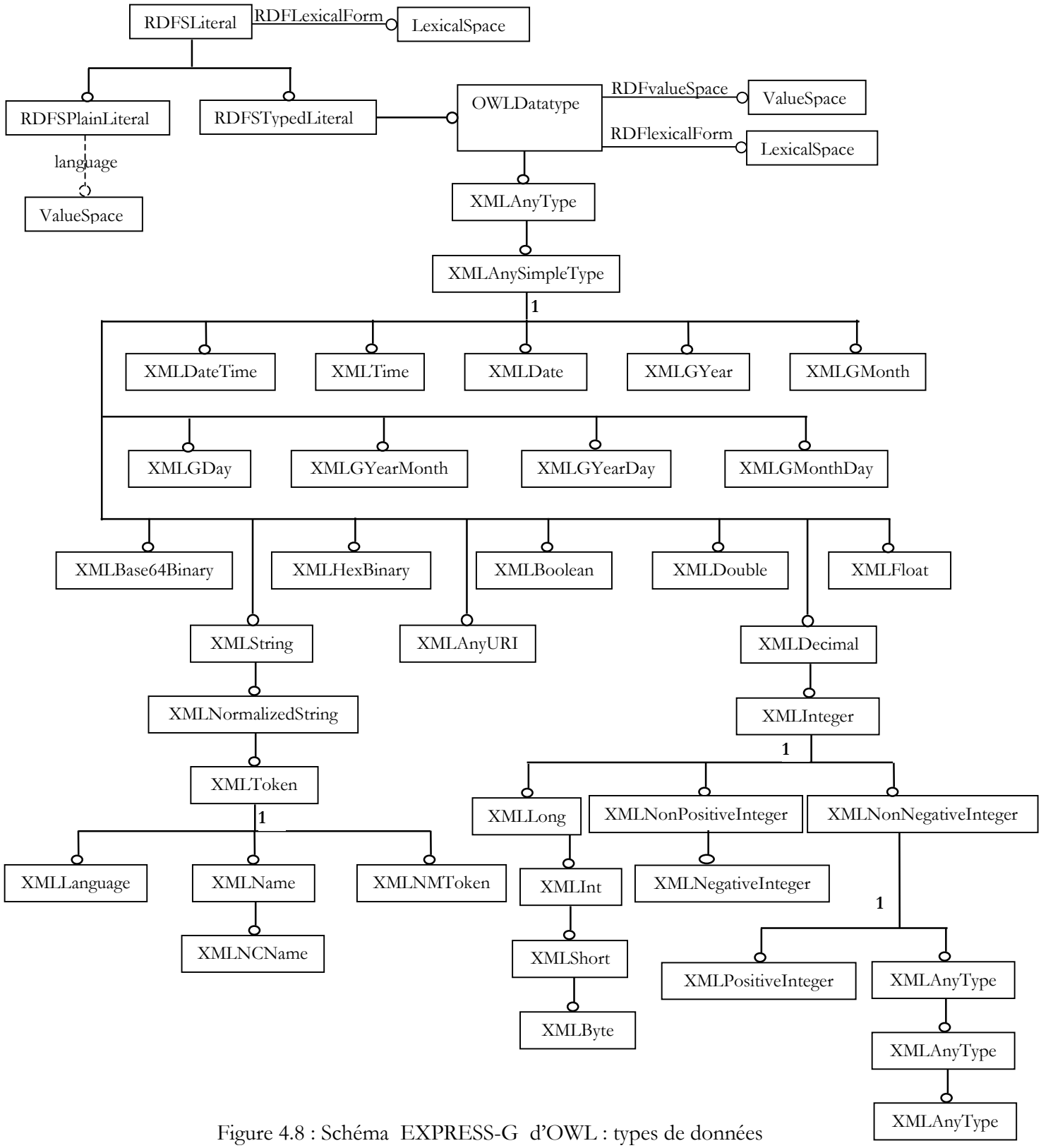


Figure 4.8 : Schéma EXPRESS-G d'OWL : types de données

Conclusion

Après avoir présenté dans cette partie la notion d'ontologie, étudié les modèles d'ontologie PLIB et OWL, nous avons été amené à la vue des méthodes de transformations de modèles, à concevoir le modèle EXPRESS d'OWL afin de représenter PLIB et OWL par un formalisme de modélisation commun.

Nous allons dans la partie suivante discuter des règles de transformations qui ont été conçues.

PARTIE II : ANALYSE ET CONCEPTION DES REGLES DE MAPPING DES MODELES PLIB ET OWL.

Introduction

La première section des chapitres V et VI de cette partie est consacré à l'identification, l'analyse et la spécification des différentes règles de transformation qui seront appliquées aux modèles PLIB et OWL.

Pour chacun des cas identifiés nous préciserons si la transformation est réalisable :

- sans modification de modèle cible,
- avec ajout de ressources au modèle cible,
- nécessite une extension du modèle cible.

Ensuite, les sections suivantes sont consacrées à la formalisation de ces règles, elles y seront présentées de manières détaillées en mettant l'accent sur la correspondance et la transformation entre les propriétés.

Dans les exemples que nous proposons, une entité sera représentée comme dans la notation EXPRESS-G, mais ses attributs (ou les types de ses attributs) pourront être omis si ces derniers n'ont pas d'intérêts dans le contexte où l'on se trouve.

CHAPITRE V : CONVERSION DE OWL VERS PLIB

Nous étudions dans ce chapitre les règles de transformations d'une ontologie OWL vers PLIB.

V.1 Analyse de la démarche de transformation

Pour mettre en œuvre cette conversion, nous allons nous baser sur les spécificités de OWL et, essayer de les transformer de façon efficace.

Nous allons axer notre analyse sur les points suivants :

1. Comment traiter les classes (classes défines, classes primitives) ?
2. Comment traiter l'héritage multiple (classes et propriétés) ?
3. Comment convertir les propriétés mathématiques des propriétés OWL (réflexive, symétrique, transitive, antysymmetric, irreflexive) ?
4. Comment représenter les classes définies (oneOf, union, intersection, complement) ?
5. Comment traiter les restrictions (cardinality, mincardinality, maxcardinality, allvaluesfrom, somevaluesfrom)?
6. Comment mapper les individus ?

Notons que nous proposerons également des règles pour transposer certains constructeurs au niveau de l'architecture OntoDB, même si ces derniers ne possèdent pas d'équivalence au niveau du modèle PLIB.

Remarque : Dans ce qui suit, si aucune hypothèse n'est faite sur une construction transposable dans le modèle PLIB, cela suppose que l'architecture OntoDB implémente déjà la relation équivalente.

V.1.1 Représentation d'une classe

PLIB définit trois catégories de classes, les classes de définition, les classes de représentation et les classes de vue.

Deux solutions peuvent être envisagées pour représenter une classe OWL.

V.1.1.1 Déterminer la catégorie de classe PLIB sémantiquement proche

Cette solution revient à faire correspondre une classe OWL à une classe de définition PLIB.item_class.

C'est une solution immédiate qui n'entraîne aucune modification du modèle PLIB.

Illustration

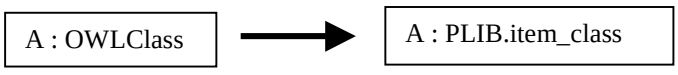


Figure 5.1a : Mise en correspondance d'une classe OWL.

V.1.1.2 Introduire une nouvelle catégorie de classe dans PLIB

Cette seconde solution consiste à introduire une nouvelle catégorie de classe dans le modèle PLIB, c'est à dire de modifier PLIB comme le montre la figure ci-dessous:

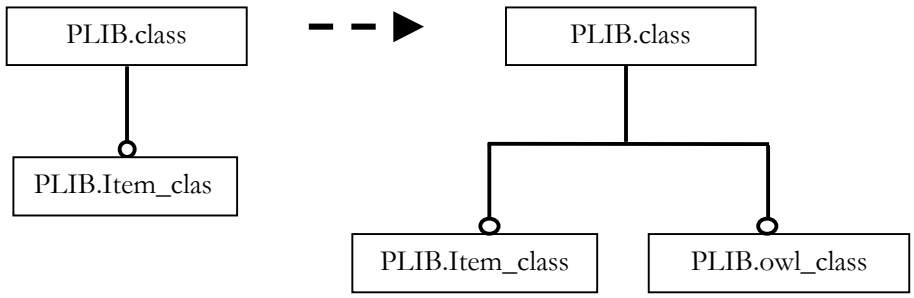


Figure 5.1b: Mise en correspondance d'une classe OWL.

Cet exemple ne représente qu'une possibilité parmi d'autres, de modification ou d'extension du modèle. Une variante de cette solution serait par exemple d'ajouter deux sous-classes à la classe PLIB.owl_class : une pour les classes primitives et une pour les classes définies, ou encore d'ajouter tout simplement à la classe PLIB.owl_class une propriété booléenne permettant de statuer sur son caractère primitif ou non.

La première solution de mise en correspondance (V.1.1.1) que nous avons présentée est une solution facile à mettre en œuvre, elle ne modifie pas le modèle PLIB, cependant, elle perd la sémantique portée par les classes OWL. Contrairement à cette solution, la seconde solution (V.1.1.2) étudiée présente l'avantage qu'elle permet de faire une distinction nette entre une classe issue du modèle OWL et une classe issue du modèle PLIB et de plus, contrairement à la première solution, elle permet également de pouvoir spécifier des propriétés propres pour les classes OWL et d'éviter la surcharge des classes OWL avec des propriétés telles que (DEPENDANT_P_DET ou encore CONDITION_DET) spécifiques à PLIB.

En addition à ces solutions, nous allons voir dans la suite que certaines classes définies seront converties différemment. Aussi leurs extensions seront traitées d'une façon particulière.

V.1.2 Cas particulier des classes définies

Quatre situations peuvent se présenter.

V.1.2.1 Classe définie par énumération de ses instances

$A \equiv \text{oneOf}(i_1, \ldots i_n)$.

a) Si les individus sont des littéraux

Alors, la classe A correspond donc en fait à un type de données énuméré et sera convertie automatiquement comme tel en PLIB

Illustration

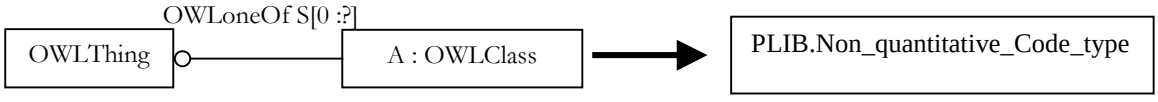


Figure 5.2 : Mise en correspondance d'une classe énumérée OWL.

Nous avons choisi de mettre en correspondance ce type de classe par un type énuméré PLIB dont l'ensemble des éléments sont représentés par une chaîne. Cette correspondance immédiate n'exploite toutefois pas toute la puissance de PLIB qui permet de créer, selon que l'ensemble des éléments soit ordonné ou pas, un type énuméré dont l'ensemble des éléments sont représentés par une valeur entière ou des chaînes. Notre choix a été guidé par le fait que OWL ne tient pas compte de ce que les éléments de l'énumération soit ordonnés ou pas.

b) Si les individus sont des instances d'une classe de l'ontologie.

Cette construction ne possède pas de correspondante au niveau du modèle PLIB. Nous proposons de transformer la classe A comme dans la section V.1.1 et de spécifier son extension par une vue. (cf. section V.1.2.3).

V.1.2.2 Classe définie comme étant le complémentaire d'une autre classe

A ≡ complementOf(B).

Ce constructeur définit l'extension d'une classe A comme étant l'ensemble des individus qui ne sont pas dans l'extension de la classe B. Au niveau ontologique, ce type de classe sera représentée comme proposé à la section V.1.1. La représentation de son extension est étudiée à la section V.1.2.5.

V.1.2.3 Classe définie comme étant une intersection

A ≡ intersectionOf(C_1 … C_n)

Nous distinguons ici deux situations principales :

a) L'intersection contient une restriction

Au niveau ontologique, la classe sera représentée comme proposé à la section V.1.1. La seule différence se situe au niveau de la représentation de son extension (instances) qui sera représentée par une vue. Cette représentation est étudiée à la section V.1.2.5.

b) L'intersection porte sur des classes primitives

L'intersection est interprétée en OWL comme la multi-instanciation, cette situation est donc sémantiquement équivalente à subClass(A, C_i) i = 1 à n. Ce cas est traité différemment du premier car les individus étant représentés uniquement dans la base de données, la vue correspondante à l'intersection des extensions des classes C_i sera toujours vide si les C_i sont des classes primitives. Nous proposons donc de mettre en correspondance ce type de construction comme dans le cas d'une classe primitive ayant plusieurs super-classes (cf. section V.1.3.2).

V.1.2.4 Classe définie comme étant une union ou une restriction

Au niveau ontologique, ces classes seront représentées comme proposé à la section V.1.1 La seule différence se situe au niveau de la représentation de leur extension (instances) qui est étudiée à la section V.1.2.5.

V.1.2.5 Extension des classes définies

En OWL, l'extension d'une classe définie est définie en appliquant des opérateurs ensemblistes. Cette construction ne possède n'a pas d'équivalence au niveau du modèle PLIB car toute instance

est définie par une unique classe de l'ontologie. Il n'est donc pas possible de représenter l'extension de ses classes dans le modèle PLIB. Cependant, au niveau de l'architecture OntoDB les extensions de ces classes vont être représentées par des « vues ».

Le choix du type de la vue (matérialisée ou non) doit être guidé par l'application. En effet pour une application orientée consultation, il serait adapté d'utiliser des vues matérialisées afin de gagner en temps d'exécution (des requêtes associées aux classes définies) et prévoir une fréquence de mises à jour en fonction de l'évolution des données.

Il convient de noter que dans la mise en œuvre de cette solution, toutes les vues ne sont pas inversibles. En effet, on ne saura pas toujours sur quel concept primitif mapper un individu dans le cas de l'instanciation dans une union.

La solution que nous avons adoptée pour représenter l'extension des classes définie offre l'avantage qu'elle permet de représenter chaque individu uniquement dans la base de données. Cette solution nécessite l'ajout d'un attribut pour le stockage de la vue correspondante à extension de la classe et donc entraîne une modification du modèle PLIB.

V.1.3 Représentation de l'héritage

Trois situations se présentent :

V.1.3.1 Classe ayant pour super classe une ou plusieurs classes nommées définies

Cette construction garde la même sémantique si on remplace la ou les classe(s) définies par leurs déclarations. On se retrouve donc dans ce que nous traitons dans la suite.

V.1.3.2 Classe ayant pour super classe une ou plusieurs classes primitives

a) Cas d'une hiérarchie simple :

Dans ce cas, la transformation est faite automatiquement.
La relation OWL SubClassOf(A, B) sera traduite en A is_a B dans PLIB.
Illustration

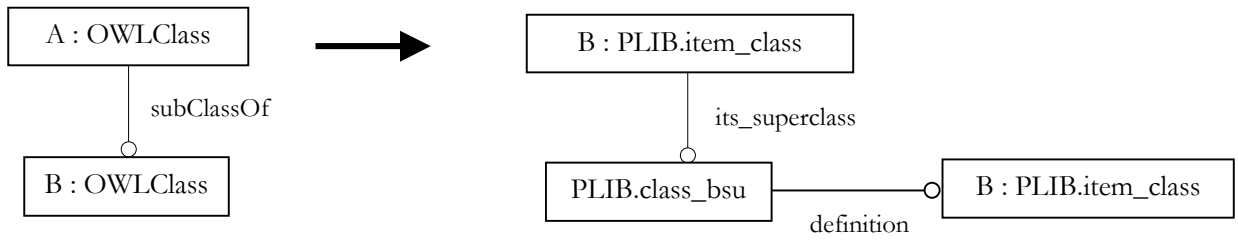


Figure 5.3 : Mise en correspondance d'une relation d'héritage simple OWL.

Cette mise en correspondance est directe et garde la même sémantique dans les modèles OWL et PLIB.

b) Cas d'une hiérarchie multiple :

La hiérarchie d'une ontologie PLIB est une hiérarchie simple, cependant, en exploitant le mécanisme "is_case_of" qui permet d'importer certaines propriétés d'une ou plusieurs classe(s), on peut simuler l'héritage multiple important toutes les propriétés des classes dites super-classes.

Toutefois, pour ne pas perdre la hiérarchie simple qui existe déjà dans PLIB (dans le cas d'un héritage simple OWL), dans le cas d'un héritage multiple une classe sera choisie pour le mécanisme "is_a" et les autres classes seront super-classes par le mécanisme "is-case_of".

Au lieu d'avoir une conversion en une PLIB.item_class comme dans la section V.1.1, nous prendrons dans ce cas une PLIB.item_class_case_of qui est une sous-classe de PLIB.item_class.

Illustration

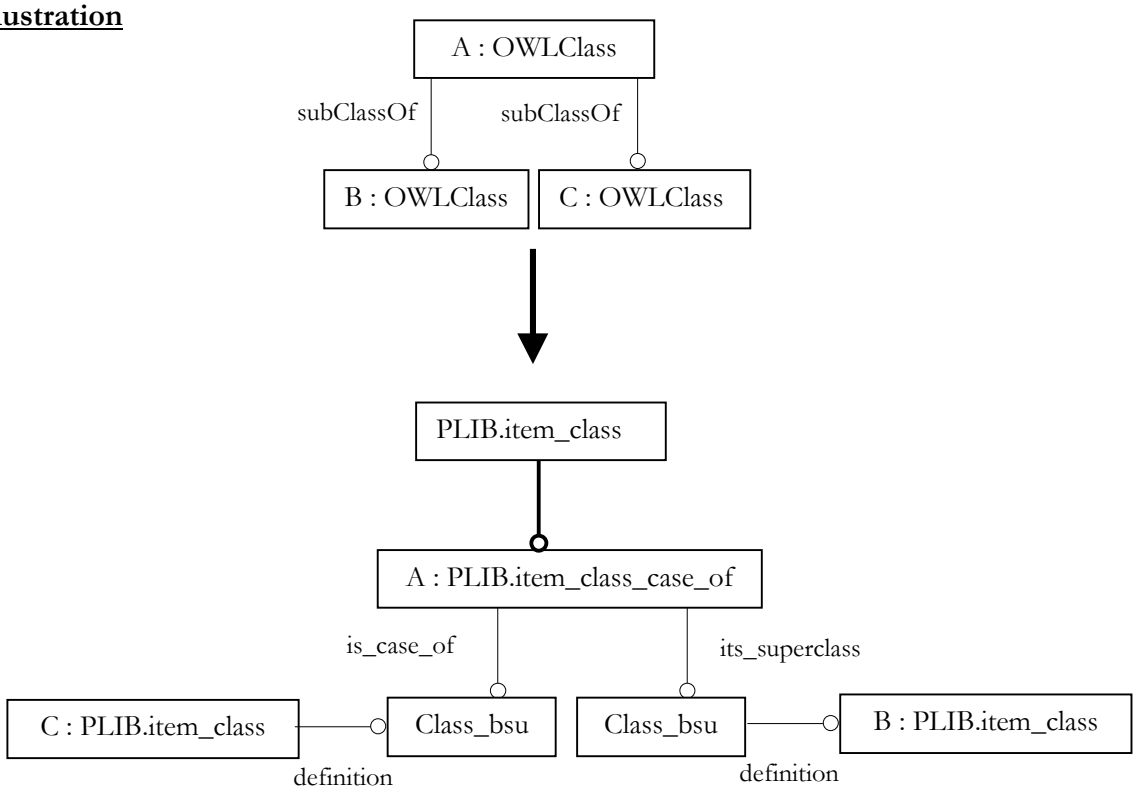


Figure 5.4 : Mise en correspondance d'une relation d'héritage multiple OWL.

Il est évident qu'une métrique doit être définie afin d'élire la meilleure classe candidate pour la relation « its_superclass » et déterminer les propriétés à importer pour les relations « is_case_of ».

Métriques de sélection de la meilleure classe candidate pour la relation « its_superclass »

- Une métrique simple et évidente serait de choisir pour la relation « its_superclass » la classe ayant le plus grand nombre de propriétés et d'importer par le mécanisme « is_case_of » toutes les propriétés des autres classes. Cependant, cette solution intuitive et facile à mettre en œuvre ne tient pas compte du fait que généralement seul un petit sous-ensemble de propriétés va être utilisé pour décrire les individus.

  Bien que la mise en correspondance dans ce cas soit donc quasi automatique et garde la sémantique de la relation initiale. L'inconvénient de cette solution est la surcharge du mécanisme is_case_of car toutes les propriétés des classes subsumantes sont importées.

- Une autre métrique serait de sélectionner la meilleure classe candidate pour la relation « its_superclass » par évaluation des instances : (choisir la classe dominante en fonction des propriétés valuées dans une instance). Etant données que les ensembles de propriétés utilisés

pour deux instances peuvent être disjoints, on va être amener à importer de nouvelles propriétés et à modifier la table d'extension chaque fois qu'une nouvelle instance est ajoutée.

Cette métrique bien qu'en conformité avec le mécanisme is_case_of sera très coûteuse au niveau de sa mise en œuvre. Notons également que cette solution ne peut être appliquée si la classe ne possède pas d'instances. C'est une solution qui entraîne une dépendance entre niveau ontologique et niveau données.

V.1.3.2 Classe ayant pour super classe une ou plusieurs classes anonymes

Cette construction sert dans certains cas à restreindre le co-domaine ou l'arité d'une propriété au niveau d'une classe.

Nous distinguons deux cas de figure principaux.

1. **Redéfinition de l'arité**

subClass(A, Restriction(P, minCardinality(n))).

Nous distinguons également deux cas de figure :

- **Si P définit A comme son domaine**

Il ne s'agit pas vraiment d'une redéfinition de P, OWL utilise ce mécanisme pour définir l'arité minimale et maximale d'une propriété. Cela sera traduit en PLIB en déclarant la propriété comme une propriété de type collection dans son domaine de visibilité A.

Illustration

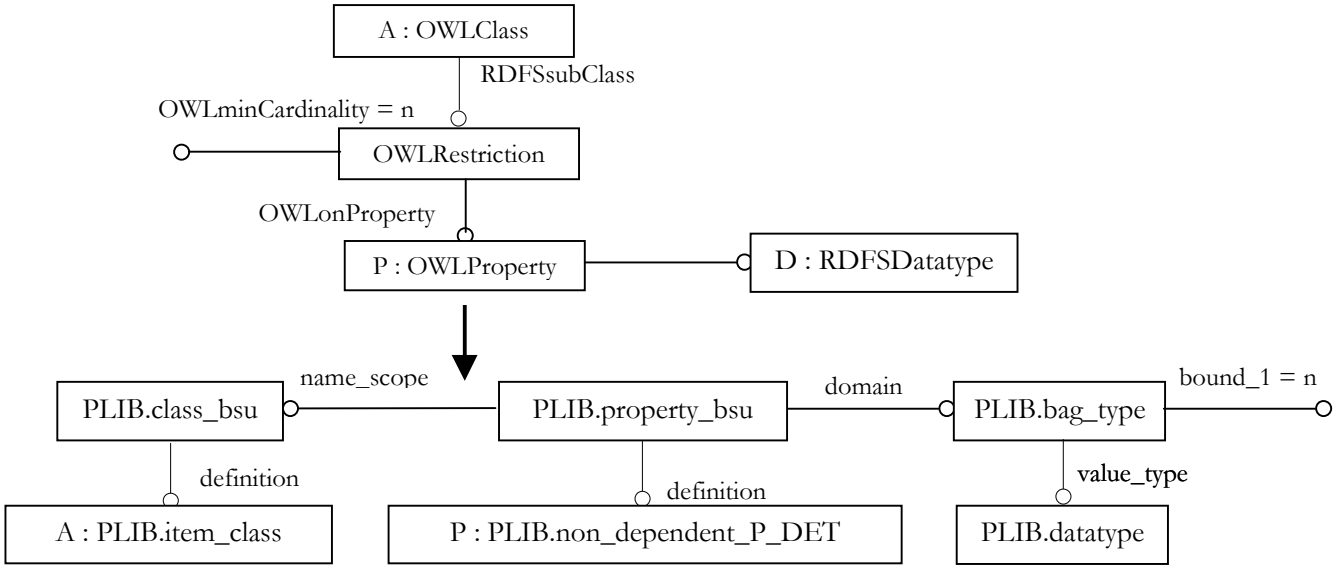


Figure 5.5 : Mise en correspondance d'une super-classe anonyme : redéfinition de l'arité.

Remarque : les conversions des propriétés et des types de données sont traitées plus loin (cf. sections V.1.5 et V.1.6).

Cette figure s'applique aussi quand OWLmincardinality = n est remplacé par OWLmaxcardinality = m ou OWLcadinality = k. Pour chacun de ses cas, il suffit de remplacer respectivement :

- bound_1 = n par bound_2 = m,
- bound_1 = n par bound_1 = k et bound_2 = k.

- **Si P ne définit pas A comme son domaine**

Il s'agit d'une redéfinition des cardinalités de la propriété au niveau de la classe A. PLIB ne fournit aucune équivalence pour ce cas. En effet, le schéma de contrainte PLIB ne permet pas de redéfinir l'arité d'une propriété.

2. Redéfinition du co-domaine

- **Cas d'une classe anonyme définie par l'axiome <owl : allvaluesFrom>**

subClass(A, Restriction(P, allValuesFrom (B)))

Si B est une classe définie

Nous ne trouvons pas d'équivalence au niveau de modèle PLIB à cette construction.

Si B est une classe primitive

Cette construction sera transformée en PLIB en ajoutant une contrainte à la classe A pour redéfinir le co-domaine de la Propriété P.

Illustration

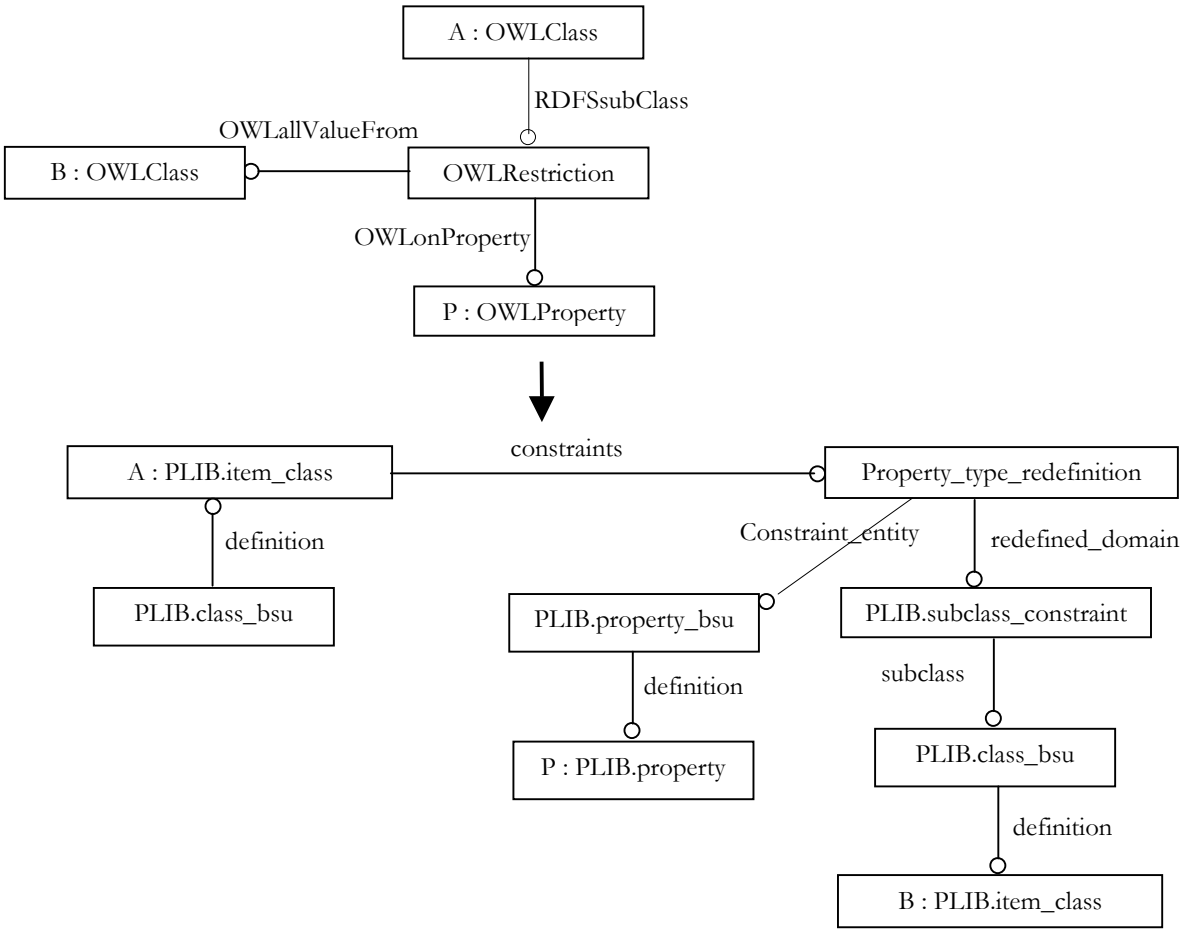


Figure 5.6 : Mise en correspondance d'une super-classe anonyme : redéfinition du co-domaine.

Si B est un ensemble de valeurs (P est de type datatypeProperty)

La figure précédente s'applique, il suffit pour cela de remplacer PLIB.subclass_constraint par PLIB.subset_constraint et d'initialiser sa propriété subset par les valeurs de B.

- **Cas d'une classe anonyme définie par l'axiome <owl : hasValue>**

subClass(A, Restriction(P, hasValue (i)))

Nous distinguons ici deux situations :

i est un littéral (P est de type datatypeproperty) :

La figure précédente s'applique, il suffit pour cela de remplacer PLIB.subclass_constraint par PLIB.string_pattern_constraint et d'initialiser sa propriété pattern par l'expression régulière « i ».

i est une instance d'une classe de l'ontologie (P est de type objectproperty):

Ce type de construction n'a pas de correspondance au niveau du modèle PLIB. Cependant, elle peut être implémentée au niveau de l'architecture OntoDB en implémentant un trigger.

- **Cas d'une classe anonyme définie par l'axiome <owl : valueNot>**

subClass(A, Restriction(P, valueNot (i)))

Nous distinguons ici deux situations :

i est un littéral (P est de type datatypeproperty) :

La figure 5.6 s'applique, il suffit pour cela de remplacer PLIB.subclass_constraint par PLIB.string_pattern_constraint et d'initialiser sa propriété pattern par l'expression régulière « !i ».

i est une instance d'une classe de l'ontologie (P est de type objectproperty):

Ce type de construction n'a pas de correspondance au niveau du modèle PLIB. Cependant, elle peut être implémenté au niveau de l'architecture OntoDB par un trigger pour vérifier cette contrainte.

- **Cas d'une classe anonyme définie par l'axiome <owl : somevaluesFrom>**

subClass(A, Restriction(P, someValueFrom(B))).

Ce type de construction n'a pas de correspondance au niveau du modèle PLIB. Cependant, elle peut être implémentée au niveau de l'architecture OntoDB par un trigger pour vérifier cette contrainte.

3. Autres

- **Cas d'une classe anonyme définie par l'axiome <owl : intersectionOf >**

subClass(A, intersectionOf(B_1, , B_n)).

Cette construction correspond à la multi-instanciation et, est sémantiquement équivalente à dire que la classe A a pour super-classe toutes les classes B_j, j :=1 à n dans l'union: On se référera donc aux règles précédentes (cf. section V.1.3).

- **Cas d'une classe anonyme définie par l'axiome <owl : unionOf >**

subClass(A, unionOf(B_1, , B_n)).

Nous ne trouvons pas d'équivalence dans le modèle PLIB dans le cas d'une classe ayant pour super-classe une classe définie pas intersection.

- **Cas d'une classe anonyme définie par l'axiome <owl : oneOf >**

subClass(A, oneOf(i_1, , i_n))

Cette construction revient à spécifier les individus qui peuvent appartenir à la classe A. Nous ne proposons pas de transformation pour cette construction.

- **Cas d'une classe anonyme définie par l'axiome <owl : complementOf >**

SubClass(A, complementOf(B))
Cette construction ne possède pas d'équivalence dans le modèle PLIB et ne peut non plus être implémentée dans l'architecture OntoDB.

V.1.4 Assertions sur les classes

V.1.4.1 Classe définie par l'axiome <owl : equivalentClass>

EquivalentClass(A, B).
Nous distinguons ici deux situations:

a) B est une classe nommée

En se situant dans un contexte OWL DL où les références circulaires ne sont pas permises,

- **Si les classes A et B sont spécifiées comme primitives**

Une classe sera choisie pour être transformée comme une classe primitive, et la seconde comme une classe définie, l'extension de cette dernière sera alors une vue.

- **Si B est une classe définie**

La classe A sera alors considérée comme une classe définie.

b) B est une classe anonyme

Il s'agit dans ce dans ce cas de la définition de la classe A, on se référera aux cas de figures discutées précédemment (cf. section V.1.1).

V.1.5 Représentation d'une propriété

Nous proposons dans cette section d'identifier et de proposer si possible des règles de transformations pour les propriétés OWL.

Comme pour les classes, PLIB définit quatre catégories de propriétés, les propriétés essentielles et indépendantes, les propriétés dépendantes, les paramètres de contexte et les propriétés non essentielles.

Deux solutions s'offrent donc à nous.

V.1.5.1 Déterminer la catégorie de propriété PLIB sémantiquement proche

Nous allons dans ce cas convertir les propriétés OWL en des propriétés essentielles indépendantes ie. PLIB.non_dependent_P_DET qui correspond à la sous-catégorie de propriété sémantiquement proche des propriétés OWL.

Illustration

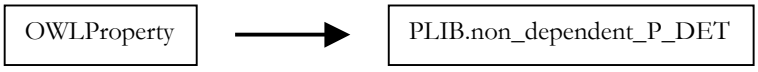


Figure 5.7a : Mise en correspondance d'une propriété OWL.

V.1.5.2 Introduire une nouvelle hiérarchie de propriété dans PLIB

Une autre solution serait comme pour les classes de définir une nouvelle hiérarchie tenant compte des caractéristiques des classes OWL, ainsi, la caractéristique (réflexive, symétrique, transitive) sera implicitement portée par l'identifiant de la propriété. Cette solution présente un grand avantage si on veut munir la base de données de capacités déductives (une autre solution pour conserver les caractéristiques des propriétés est discutée à la section V.1.9.4).

Illustration

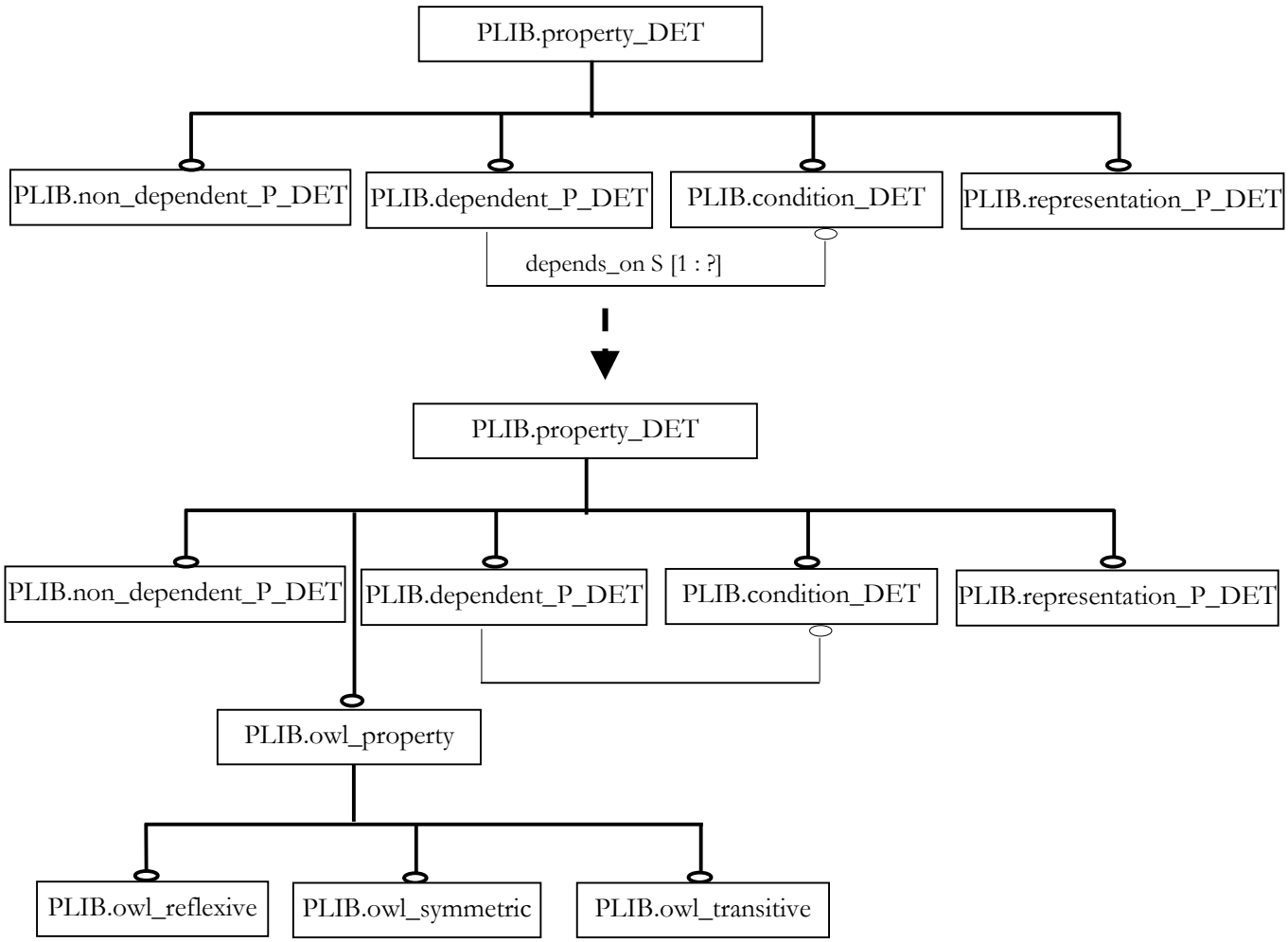


Figure 5.7b : Mise en correspondance d'une propriété OWL.

La première solution de mise en correspondance (V.1.5.1) que nous avons présentée est une solution facile à mettre en œuvre. Elle ne modifie pas le modèle PLIB, cependant, elle ne permet pas de conserver les caractéristiques des propriétés OWL (symétrique, réflexive,…). Contrairement à cette solution, la seconde solution (V.1.5.2) étudiée bien que modifiant le modèle PLIB présente l'avantage qu'elle permet d'associer au niveau ontologique un comportement aux propriétés.

V.1.6 Prise en compte des catégories de propriétés OWL

OWL possède deux types de constructeurs de propriété de base selon que le co-domaine d'une propriété soit une classe de l'ontologie ou un type primitif.

V.1.6.1 Cas d'une propriété objet

C'est une propriété ayant pour co-domaine une classe de l'ontologie. Ce type de propriétés sera converti en PLIB en déclarant co-domaine comme étant de type PLIB.class_instance_type.

Illustration

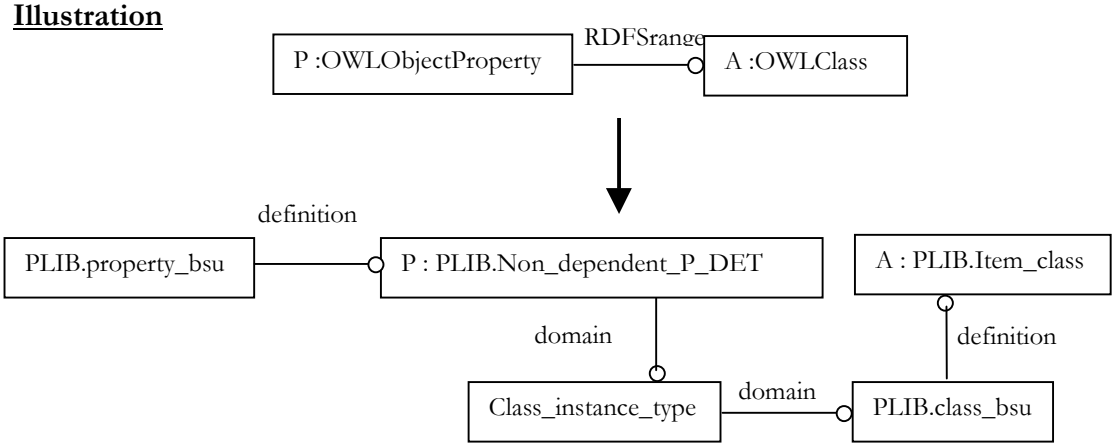


Figure 5.8 : Mise en correspondance d'une propriété objet OWL.

V.1.6.1 Cas d'une propriété simple

C'est une propriété ayant pour co-domaine un type primitif XML Schema. Ce type de propriété sera converti en PLIB en déclarant co-domaine comme étant de type PLIB.simple_type (cf. section V.1.10 pour la correspondance entre les types primitifs OWL et PLIB).

Illustration

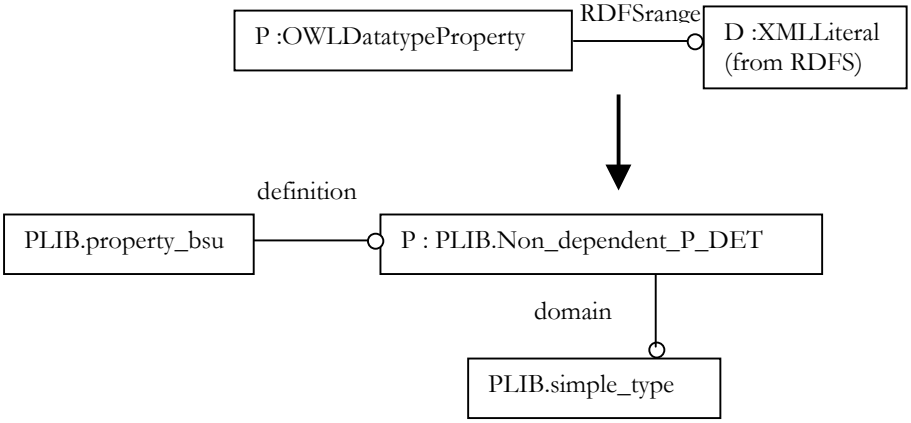


Figure 5.9 : Mise en correspondance d'une propriété simple OWL.

V.1.7 Représentation des sous propriétés

Le lien d'héritage entre les propriétés apporte une information qui serait utile dans la re-écriture des requêtes par exemple. Nous ne proposons pas au niveau ontologique un moyen de garder ce

lien. Cependant on pourrait dans OntoDB garder cette relation dans une table grâce à un attribut « subPropertyOf » comme pour les autres caractéristiques des propriétés (cf section V.1.9.4).

V.1.8 Conversion du domaine et du co-domaine d'une propriété

V.1.8.1 Cas d'une propriété ayant comme domaine ou co-domaine implicite owlThing

Le problème des propriétés ne possédant pas de domaine ou de co-domaine doit aussi être résolu lors du passage de OWL à PLIB car dans ce dernier une propriété est obligatoirement associé à une classe qui représente son domaine de visibilité.

Dans OWL, une propriété dont le domaine (respectivement le co-domaine) n'est pas spécifié est interprétée comme une propriété visible par toutes les classes de l'ontologie, c'est-à-dire une propriété ayant comme domaine (respectivement co-domaime) implicite la classe OWLThing. Cette construction est fort utile mais n'a de sens qu'au niveau d'une ontologie donnée et indépendante. En effet, une propriété P ayant un sens dans toutes les classes d'une ontologie A, n'en aura certainement pas dans pour les classes d'une autre ontologie B référençant l'ontologie A. Ce constat nous a fait penser à introduire une nouvelle classe, super classe de toutes les classes de l'ontologie considérée et qui sera implicitement le domaine (respectivement le co-domaine dans le cas des propriété de type objectProperty) de toute propriété ayant comme domaine implicite ou explicite la classe OWLThing.

Remarque : *La super-classe introduite aura pour nom le namespace de l'ontologie considérée. Dans la suite nous utiliserons la lettre « O » pour faire référence à cette classe. Notons qu'il y aura autant de super-classes « O » que d'ontologies mises en correspondances.*

Remarquons également que l'introduction de la super-classe « O » ne modifie pas sémantique des informations de l'ontologie OWL.

Cette solution modifie par contre certaines des règles de transformation des classes précédentes comme suit : une classe primitive ne possédant pas de super-classe deviendra une sous-classe de la super-classe « O ».

V.1.8.2 Cas d'une propriété ayant un ou plusieurs domaines explicites

domain(A_1), …, domain(A_n)

OWL interprète le domaine (respectivement le co-domaine) d'une propriété comme l'intersection des classes A_1, ..., A_n. Ainsi, cette construction sera traduite en spécifiant comme domaine de visibilité de la propriété la classe A_j, $j \in \{1..n\}$ si elle existe équivalente à l'intersection des A_i, i=1à n, sinon, dans le cas ou l'intersection est vide, la propriété ne sera pas représentée. Notons aussi que la propriété sera applicable au niveau de la classe A_j et de toutes ses sous-classes.

Illustration

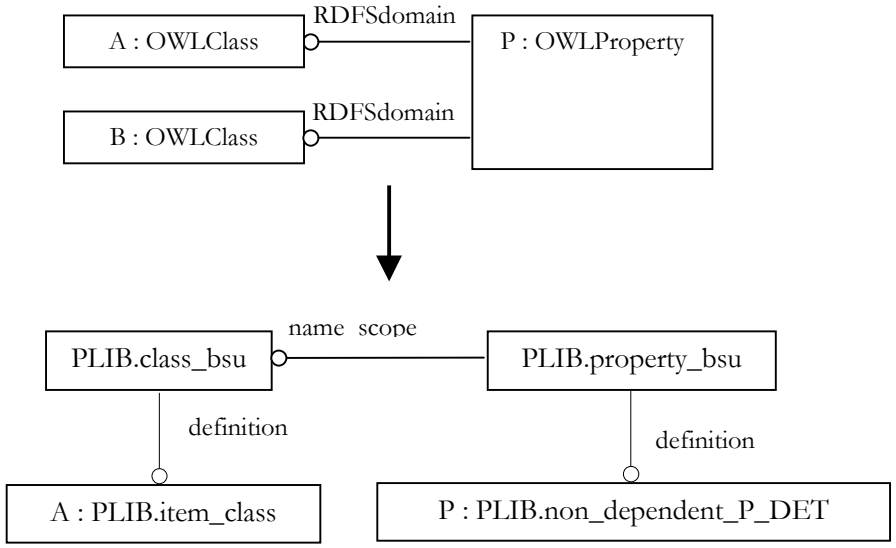


Figure 5.10 : Mise en correspondance du domaine d'une propriété OWL

Nous supposons ici que l'intersection de A est B est la classe A.

Remarque : dans tous les cas, la propriété est visible et applicable dan son domaine de visibilité.

V.1.8.3 Cas d'une propriété ayant un ou plusieurs co-domaines

Cette construction sera traduite en spécifiant comme co-domaine de visibilité de la propriété la classe A_j $j \in \{1..n\}$ si elle existe équivalente à l'intersection des A_i, i=1à n, sinon, dans le cas ou l'intersection est vide, la propriété ne sera pas représentée.

Illustration.

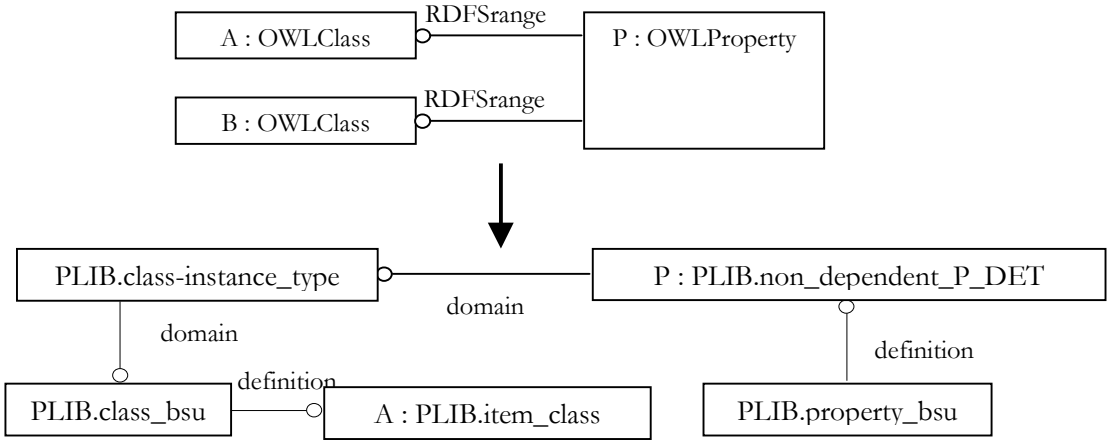


Figure 5.11 : Mise en correspondance du co-domaine d'une propriété OWL

Nous supposons ici que l'intersection de A est B est la classe A.

V.1.9 Représentation des caractéristiques des propriétés

V.1.9.1 Cas d'une propriété fonctionnelle

Cette caractéristique permet de spécifier qu'une propriété ne peut avoir plus d'une valeur. Cette caractéristique est sémantiquement équivalente à dire la classe domaine de la propriété à pour super-classe la classe anonyme définie par une restriction de cardinalité maximale 1 sur cette propriété, ce qui nous renvoie à la section V.1.3.2.

Illustration

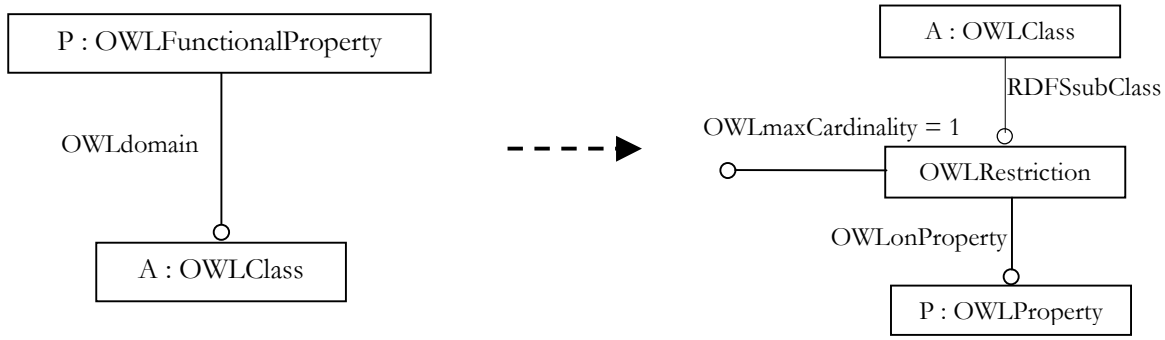


Figure 5.12 : Mise en correspondance d'une propriété fonctionnelle OWL.

Notons que le formalisme EXPRESS permet de définir une propriété comme étant optionnelle grâce au mot clé OPTIONAL, mais ce mécanisme n'est pas explicitement intégré dans OntoDB.

Remarque : si aucune précision n'est faite sur le caractère fonctionnel d'une propriété, alors celle-ci sera transformée en PLIB en une propriété de type agrégat (cf. section V.1.2.3) au niveau de sa classe de visibilité.

V.1.9.2 Cas d'une propriété inverse fonctionnelle

Bien que formalisme EXPRESS propose le mécanisme UNIQUE pour exprimer le fait qu'une propriété est une fonction injective, cette modélisation n'est pas intégrée à OntoDB. Cependant, cette caractéristique y est implicitement implémentée en transformant de telles propriétés en clés primaires.

V.1.9.3 Cas d'une propriété réflexive, symétrique ou transitive

Ces propriétés seront traitées dans un premier temps au niveau de modèle PLIB sans tenir compte de ses caractéristiques car le modèle PLIB ne propose pas de mécanisme pour les supporter. Il est cependant, évident que la connaissance du caractère réflexif, symétrique ou transitif d'une propriété pourrait être d'une incidence importante pour les applications, par exemple pour faciliter la création d'individus et maintenir automatiquement les contraintes qu'elles représentent. Elle pourrait également être prise en compte dans la réécriture des requêtes.

On pourrait par exemple envisager dans le cas d'une propriété symétrique ou réflexive, si on conserve ces informations de saturer la base de données à l'aide d'un trigger car cette opération sera peu coûteuse.

- Pour une propriété réflexive, sa valeur n'aura pas besoin d'être fournie, elle sera automatiquement renseignée. Il faudra également prévoir un trigger afin d'empêcher toute modification ou suppression de la relation symbolisée par cette propriété.
- Pour une propriété symétrique, à l'insertion ou à la suppression de tout couple (x,y) un trigger insèrera ou supprimera également le couple (y,x).

C'est fort de ces exemples que nous avons proposé pour conserver ces caractéristiques essentielles d'introduire au niveau d'OntoDB une table pour stocker ces informations. (cf. section suivante V.1.9.4).

Pour les propriétés transitives, on peut envisager deux types de saturations :

- la saturation à l'insertion qui n'est pas inversible. En effet, la présence des couples (x,y) et (y,z), déclenchera l'ajout d'un couple (x,y) , cependant, en cas de suppression d'un ou des deux couples (x,y) et (y,z), on ne sait pas à priori s'il y a obligation de supprimer le couple (x,z) à moins d'avoir gardé une trace de tous les couples ajoutés lors du déclenchement de la saturation ce qui représente une gestion "lourde".
- la saturation à la consultation : dans cette seconde solution, les couples inférés ne seront pas insérés dans la base de données mais stocké dans un emplacement temporaire dans le but de répondre aux besoins de visualisation des données de l'utilisateur. Cette solution offre l'avantage ne pas poser de problèmes lors de la suppression d'un couple.

Dans tous les cas, le choix du type de saturation et du moment auquel il doit être réalisé doit être guidé par l'application car il représente un coût non négligeable tant en terme de temps de traitement qu'en terme de gestion des données. On va par exemple privilégier la saturation à l'insertion pour une application de type entrepôt de données afin de gagner en temps de réponse lors des consultations.

V.1.9.4 Cas d'une propriété ayant une propriété inverse

La caractéristique INVERSE bien que spécifiée par EXPRESS n'est pas supportée par PLIB. Nous pouvons choisir de représenter à la fois, au niveau du modèle d'ontologie PLIB, les propriétés et leurs inverses. Il est cependant évident que, comme pour le cas des propriétés transitives, la connaissance du caractère inverse d'une propriété pourrait être d'une incidence importante pour les applications ; par exemple elle va permettre d'avoir un double accès aux données (la question : est ce que Paul « est le fils de » Pierre, pourra être reformulée en : est ce que Pierre « est le père de » Paul.

Les caractéristiques des propriétés seront stockées au niveau d'OntoDB par la table suivante :

property_bsu	isReflexive	isSymmetric	isTransitive	inverseOf	subPropertyOf
P001	boolean	boolean	boolean	P002	P002

Tableau 5.1 : Stockage des caractéristique des propriétés OWL

La mise en œuvre pour le cas de propriété inverse est différente de celles des propriétés symétriques ou transitives. En effet, dans le cas de deux propriétés inverses l'une de l'autre, l'implémentation au niveau de l'architecture OntoDB doit tenir compte des dépendances fonctionnelles :

a) Cas d'une dépendance 1 à 1

Dans ce cas, les propriétés sont fonctionnelles symétriques dans les deux sens. Elles seront toutes deux représentées mais une seule propriété pourra être fournie lors de l'ajout d'une instance, et la seconde sera alors automatiquement saturée.

b) Cas d'une dépendance 1 à n

Dans ce cas, seule la propriété fonctionnelle sera représentée.

c) Cas d'une dépendance n à m

Dans ce dernier cas, les deux propriétés ne sont fonctionnelles dans aucun sens. Une table intermédiaire où figureront les deux propriétés sera crée pour matérialiser la relation.

V.1.10 Prise en compte de l'axiome disjointProperties

Cet axiome ne possède pas de correspondance dans le modèle PLIB, bien qu'il soit important. Il permet de spécifier qu'un individu ne peut pas jouer deux rôles à la fois (à titre d'exemple, les propriétés aPourEpouse et aPour_Enfant doivent être disjointe pour la classe HOMME).

V.1.11 Conversion des types de données

Trois situations se présentent :

V.1.11.1 Cas des types de données primitifs

Le tableau ci-dessous établit la correspondance entre les types de données primitifs recommandés par OWL et les types de données PLIB.

OWL	PLIB
xsd:time, xsd:date, xsd:gYearMonth, xsd:gYear, xsd:gMonthDay, xsd:gDay, xsd:gMonth	Entity_instance_type(« date »)
xsd:anyURI,	remote_http_address
xsd:boolean	boolean_type
xsd:decimal	int_type
xsd:double ,xsd:float	real_type
xsd:integer, xsd:nonNegativeInteger, xsd:positiveInteger	integer_type
xsd:language	language_code
xsd:long, xsd:int, xsd:short, xsd:byte, xsd:unsignedLong, xsd:unsignedInt, xsd:unsignedShort, xsd:unsignedByte	integer_type
xsd :string, xsd:normalizedString, xsd:token xsd:NMTOKEN,	string_type

Tableau 5.2 : Correspondance entre types simples OWL et PLIB

Remarque : cette mise en relation n'est pas inversible ; en effet, étant donné une valeur de type real_type par exemple, on ne saurait dire si elle doit être reconvertie en une valeur de type xsd :float ou en une valeur de type xsd :double.

V.1.11.2 Cas des types de données énumérées

Ils seront transformés soit en des types énumérés PLIB (PLIB.non_quantitative_int_type, ou PLIB.non_quantitative_code_type, soit en selon qu'ils représentent une énumération d'entiers ou chaîne de caractères).

Illustration

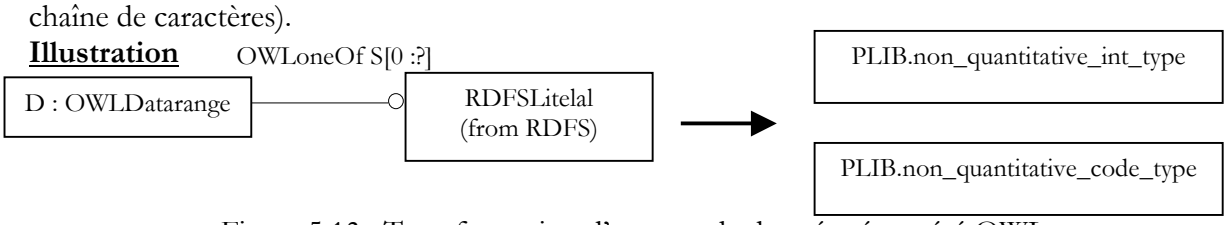


Figure 5.13 : Transformation d'un type de données énuméré OWL

V.1.11.3 Cas des types de données définis par les utilisateurs

La définition de types de données personnalisés dans OWL consiste à appliquer une ou plusieurs facettes ou contraintes sur un autre type de données.

PLIB permet uniquement de définir des contraintes sur un type de données dans le contexte d'une classe. Nous proposons donc de spécifier la super-classe « O». comme domaine de visibilité de tout type de donnée OWL (cf. section V.1.8.1), ce qui va nous permettre d'exploiter le schéma des contraintes PLIB pour la conversion des types de données personnalisés OWL en des types de données PLIB. Cette transformation va s'effectuer en deux étapes : tout d'abord la transformation en un PLIB.named_type de domaine de visibilité la super-classe « O », et ensuite la définition de la contrainte sur la super-classe « O ».

a) Cas d'une contrainte portant sur la longueur du type de données de base

Illustration

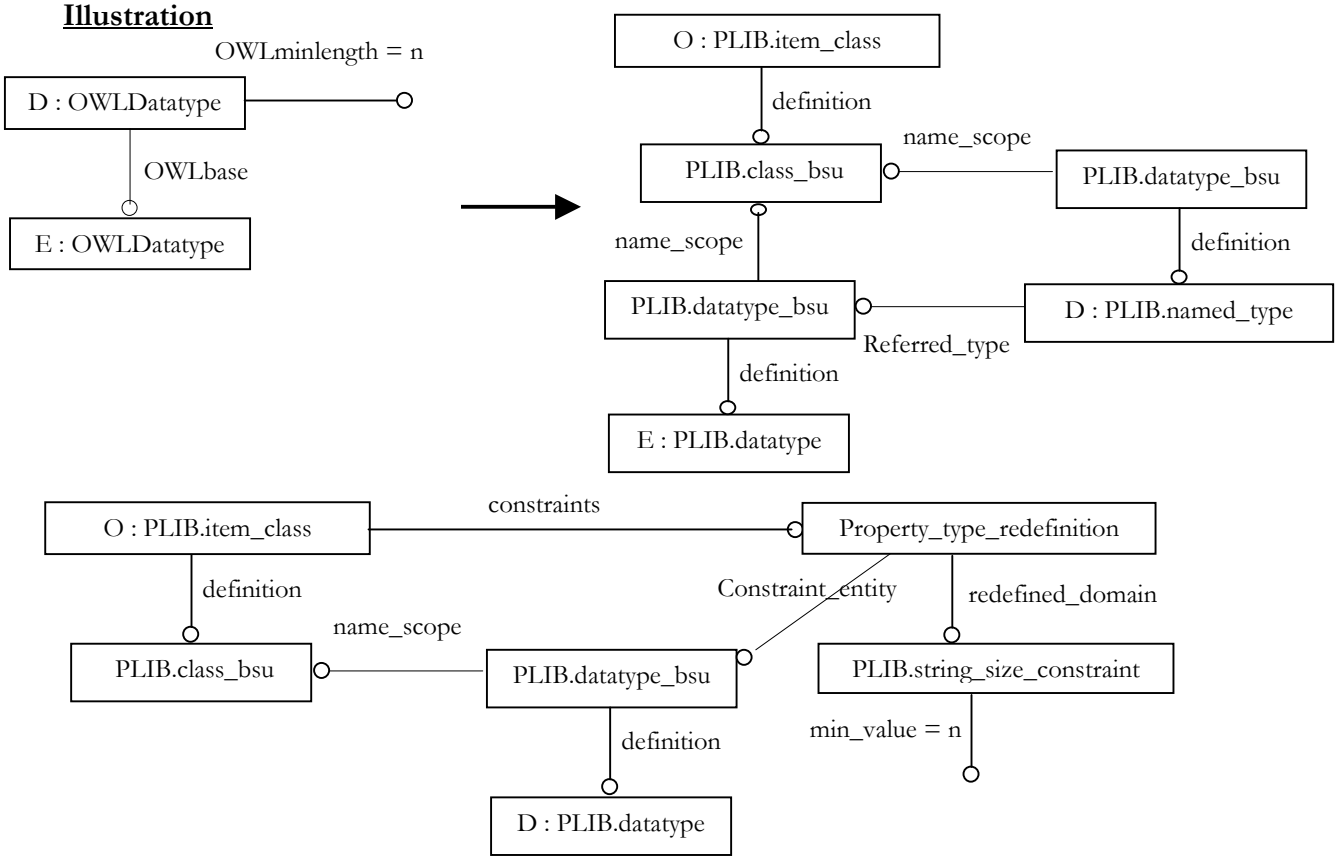


Figure 5.14 : Transformation d'un type définit OWL : contrainte de longueur

Remarques : les autres facettes de contraintes de types de données seront traitées de façon identiques.

- les cas OWLmaxLength = m et OWLength = k, reviennent à spécifier respectivement en lieu et place de min_value =n, max_value = m et min_value =k, max_value =k.

- le cas OWLpattern = «une_expression_regulière », correspond à une contrainte PLIB.string_pattern_constaint où la propriété pattern est initialisée à «une_expression_regulière ».

- le cas OWLenumération = {i_1, …, i_n} », correspond à une contrainte PLIB.subset_constaint où la propriété subset est initialisée à {i_1, …, i_n}.

- les cas OWLminInclusive = n et OWLmaxInclusive = m, correspondent respectivement à une contrainte PLIB.range_constaint où lieu min_value =n, max_value = n et max_value =m.

- les cas OWLminExclusive = n et OWLmaxExclusive = m, correspondent respectivement à une contrainte PLIB.range_constaint où min_value =n+1, et max_value =m+1.

Après avoir présenté les différentes relations que nous souhaitons maintenir et les solutions possibles pour chacune de ces relations, nous présentons dans la suite le principe de mise en correspondance qui va permettre de conserver les informations utiles citées dans les différents cas dans cette section. Cela va nous permettre de formaliser les solutions retenues (il s'agit de celles qui peuvent être réalisées sans modification ou extension du modèle cible pour les cas étudiés).

V.2 Spécification du mapping d'une ontologie OWL vers le modèle PLIB

V.2.1 Principes de transformations

La section précédente (section V.1) nous a permis de mieux analyser les problèmes auxquels nous sommes confrontés. Nous allons dans cette section spécifier les différentes règles de transformation du modèle OWL que nous avons retenues.

- Il sera introduit une super-classe identifiée par le namespace de chaque ontologie.
- Toute classe OWL sera convertie en une classe de définition PLIB.
- La transformée de toute classe OWL sera une sous-classe de la super-classe introduite.
- La transformée de toute classe OWL possédant plusieurs super-classes primitives sera sous-classe de la transformée de la super classe ayant le plus grand nombre de propriétés et importera par le mécanisme is_case_of toutes les propriétés des autres super-classes.
- Le critère de choix de la classe dominante dans le cas de la transformation d'une classe ayant plusieurs super-classes sera basé sur le nombre de propriétés.
- L'extension de toute classe définie ne sera pas représentée dans PLIB, elle sera calculée à l'aide d'une vue.

- La transformée de toute classe OWL possédant une super-classe anonyme de type restriction possèdera une contrainte correspondante à cette contrainte si elle peut être traduite dans le modèle PLIB.
- Toute classe OWL possédant une super-classe anonyme ou définie de type intersection sera interprétée comme sous-classe de chacune des classes dans l'intersection.
- La transformée de toute classe OWL définie par énumération de ses instances et portant sur des littéraux sera un type de donnés énuméré.
- Toute classe OWL définie par énumération de ses instances et portant sur des individus d'une autre classe sera transformée comme une classe définie et une vue permettra de calculer ses instances.
- Toute propriété sera transformée en une propriété essentielle PLIB, elle sera visible et applicable sur tout son domaine de visibilité
- Toute propriété sera transformée en une propriété de type agrégat, si une propriété est fonctionnelle alors l'arité maximale sera fixée à 1.
- Toute propriété ne spécifiant pas son domaine ou son co-domaine aura pour domaine de visibilité ou co-domaine la super-classe introduite.
- Les caractéristiques (réflexive, symétrique, transitive, inverse) des propriétés seront conservées par une table dans la base de données.
- Tout type de données aura comme domaine de visibilité la super-classe introduite.

Les règles que nous venons de retenir peuvent être combinées pour une entité donnée.

V.2.2 Règles de transformation des propriétés d'annotations

Dans la section V.1, nous avons principalement mis l'accent sur les relations entre entités du modèle source OWL et les relations entre entités du modèle cible PLIB. Toutefois, la conversion de modèles affecte aussi les méta-descripteurs qui sont essentiels pour les ontologies.

Nous proposons d'établir la correspondance suivante entre les méta-descripteurs de OWL et celles de PLIB.

- Tous les « premiers » labels de chaque langue (RDFSlabel($lang_i$, val_1), i = 1 à n) seront mis en correspondance avec la propriété preferred_name de la classe PLIB.item_names;
- Les autres (RDFSlabel($lang_i$, val_j), i = 1 à n, j > 1) seront mis en correspondance avec la propriété synonymous_name de la classe PLIB.item_names;
- Tous les « premiers » commentaires de chaque langue (RDFScomment($lang_i$, val_1), i = 1 à n) mis en correspondance avec sur la propriété PLIB définition ;
- Les autres (RDFScomment($lang_i$, val_j), i = 1 à n, j > 1) seront mis en correspondance avec la propriété PLIB note.

Remarque : Cet algorithme est non déterministe ; à l'issu de deux interrogations identiques avec le raisonneur JENA, Les méta-descripteurs apparaissent dans deux ordres différents.

V.2.2 Règles de calcul du domaine et du co-domaine d'une propriété

Les caractéristiques des propriétés OWL vont servir à calculer le domaine et le co-domaine de certaines propriétés lorsque ces derniers ne sont pas explicitement fournis. Nous avons établi les règles suivantes :

- si une propriété est réflexive, symétrique ou transitive, alors son domaine et son co-domaine sont égaux ;
- le domaine d'une sous-propriété est celui de sa super-propriété ;
- si le co-domaine d'une sous-propriété n'est pas spécifié alors, prendre celui de sa super-propriété ;
- le domaine (respectivement le co-domaine) d'une propriété P est le co-domaine (respectivement le co-domaine) de la propriété inverse de P et inversement.

V.3 Mise en œuvre de la correspondance d'une ontologie OWL vers PLIB

Le langage EXPRESS possède un langage de mapping : EXPRESS-X que nous présentons en section VI.4.3. Cependant, EXPRESS-X ne sera pas utilisé pour le cas du passage de OWL vers PLIB car nous nous sommes basé, pour illustrer les différentes solutions que nous avons envisagées, sur un modèle EXPRESS de OWL. Ce modèle source n'est toutefois pas un modèle effectif. Les relations que nous avons établies dans ce modèle sont en réalité reconstituées par des raisonneurs (RACER, JENA) sur des fichiers d'ontologies OWL.

Nous avons donc été amener à proposer un algorithme de mapping de OWL vers PLIB. Une autre solution aurait été d'utiliser EXPRESS-X ; cependant cette solution nécessitait auparavant d'écrire un algorithme afin de peupler automatiquement le schéma EXPRESS correspondant à OWL à partir d'une ontologie OWL donnée. Et ensuite, d'écrire le schéma de mapping EXPRESS-X ; ce qui représentait pour nous une charge de travail assez lourde. Nous avons donc opté pour l'élaboration d'un algorithme de mapping direct de OWL vers PLIB.

L'algorithme que nous avons proposé est constitué de quatre modules principaux.

- Un module de mapping des classes
- Un module de mapping de propriétés
- Un module de mapping des types de données
- Un module de mapping des instances

Notons que ces 4 modules sont étroitement liés. En effet :

- le mapping des propriétés ne peut se faire qu'une fois les classes déjà mises en correspondances,
- les types de données sont recensées et mises en correspondances lors du des propriétés, et,
- le mapping des instances n'est possible qu'une fois les trois premiers effectuées.

Les différents modules identifiés ont été partiellement implémentés par un autre stagiaire du LISI. A partir d'un fichier .owl contenant une ontologie OWL, l'algorithme implémenté par ce stagiaire retourne l'ontologie PLIB correspondante dans un fichier .spf .

Cet algorithme a été mis en œuvre en utilisant :

1. PLib API(http://plib.ensma.fr) : cette API est une API de communication avec la base de données ECCO qui fournit un ensemble de primitives permettant de modifier le contenu de la base de données, c'est-à-dire de réaliser des mises à jour, de créer ou détruire des classes de l'ontologie et leur contenu,
2. Jena API(http://jena.sourceforge.net) : c'est une API qui et utilisée pour créer, manipuler et interroger des graphes RDF. Elle permet aussi de :
    - lire et écrire du RDF et du XML
    - naviguer sur un graphe
    - interroger un graphe

Les correspondances implémentées sont :

- Classe primitive sans super-classe ;
- Classe primitive avec super-classe de type classe primitive ;
- Classe définie par énumération ;
- Propriété sans domaine et/ou co-domaine multiple et sans prise ;
- Types de données simples ;
- Instances de classes primitives ;
- Méta-descripteurs.

Ce mapping est partiel et nécessite plusieurs révisions avant l'obtention un mapping total de OWL vers PLIB.

CHAPITRE VI : CONVERSION DE PLIB VERS OWL.

A la suite de l'étude de modèle PLIB que nous avons effectuée au chapitre II, il apparaît que le modèle PLIB est un complexe. Il est constitué de 179 entités. Cependant, nous remarquons dans l'usage courant que dans les ontologies construites suivant le modèle PLIB, seul un sous-ensemble privilégié de ses entités est utilisé.

VI.1 Critère de sélection des entités à convertir

Dans le but de factoriser autant que possible les propriétés (cf. section II.4.3), il a été introduit dans le modèle PLIB un certain nombre de classes abstraites. Ces classes n'étant pas destinées à être instanciées, nous ne discuterons dans ce chapitre que des classes « feuilles » c'est à dire, celles pouvant être instanciées.

VI.2 Analyse de la démarche de transformation

Pour mettre en œuvre cette conversion du modèle PLIB vers le modèle OWL, nous allons nous baser sur les spécificités de PLIB et, essayer de les transformer de façon efficace. Nous allons axer notre analyse sur les points suivants :

1. Comment transposer les catégories de classes (item_class, functional_model_class, functional_view_class) ?
2. Comment représenter l'héritage et l'importation ?
3. Comment conserver la sémantique des informations des propriétés d'annotations de PLIB (preferred_name, definition, remark, …) ?
4. Comment transposer les catégories de propriétés PLIB (non_dependent_P_DET, dependent_P_DET) ?
5. Comment transposer le système de types de PLIB (measure_type, currency_type, quantitative_code_type) ?
6. Comment transposer le système de contrainte de PLIB?

Lors de notre étude, il s'est avéré que certaines relations définies dans le modèle PLIB ne possédaient pas d'équivalence sémantique ou de concepts ou relations sémantiquement proches dans le modèle OWL. Pour ces cas de figure, la solution que nous avons proposée se fonde sur les travaux en cours visant à étendre la sémantique du langage OWL. Ces solutions seront discutées à la fin de cette étude.

VI.2.1 Représentation d'une classe

Trois cas peuvent se présenter.

VI.2.1.1 Cas d'une classe de définition

Nous proposons de transposer cette catégorie de classe PLIB en une classe OWL.

Illustration

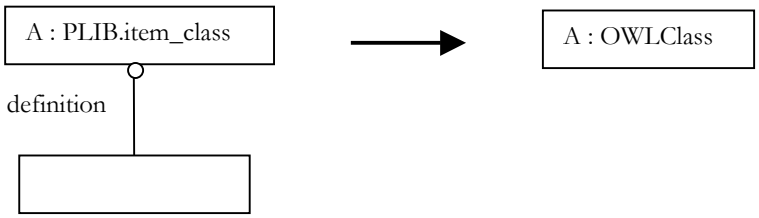


Figure 6.1 : Mise en correspondance d'une classe de définition PLIB

VI.2.1.2 Cas d'une classe de représentation

Cette catégorie de classe sera également transposée en une OWLClass, nous proposons afin de faire la différence avec les classes de définition, d'adjoindre une annotation initialisée à la chaîne « FUNCTIONAL_MODEL_CLASS » en majuscule.

Illustration

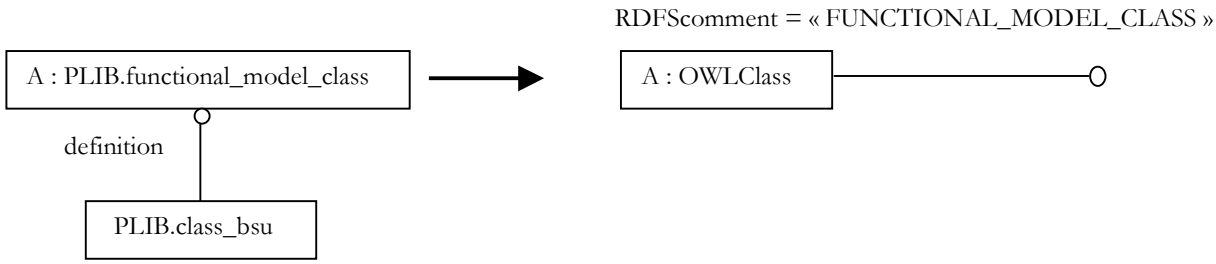


Figure 6.2 : Mise en correspondance d'une classe de représentation PLIB

Remarque : dans le cas ou la classe considérée est marquée comme désapprouvée, cette information est prise en compte en convertissant la classe en OWLDeprecatedClass qui est une sous-classe de OWLClass.

Illustration

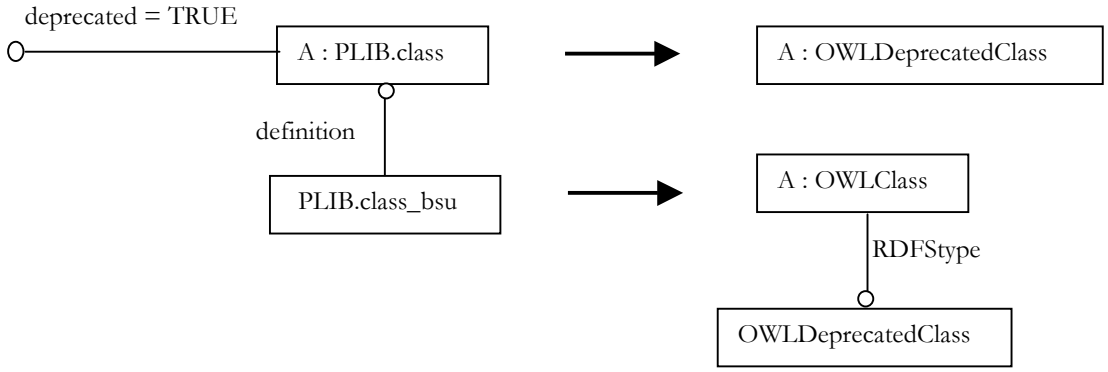


Figure 6.3 : Mise en correspondance d'une classe PLIB désapprouvée

VI.2.1.2 Cas d'une classe de vue

Ce cas sera traité à la section VI.3 car la solution que nous proposons nécessite une extension du langage OWL.

VI.2.2 Représentation de l'héritage

La hiérarchie de classe PLIB étant une hiérarchie simple, la conversion vers OWL est automatique.

Illustration

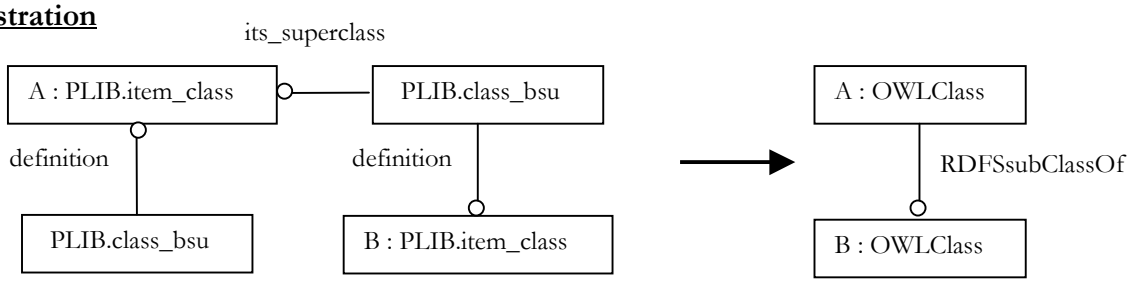


Figure 6.4 : Mise en correspondance d'une relation d'héritage PLIB

VI.2.3 Représentation de l'importation

Ce mécanisme bien que ne traduisant pas l'héritage au sens de la modélisation objet sera traduit comme tel. En effet, dans une relation « is-case_of », seul un sous–ensemble de propriété est importé. La solution que nous proposons entraîne donc une perte de sémantique de la relation de départ mais offre l'avantage de ne pas modifier le modèle OWL.

Illustration

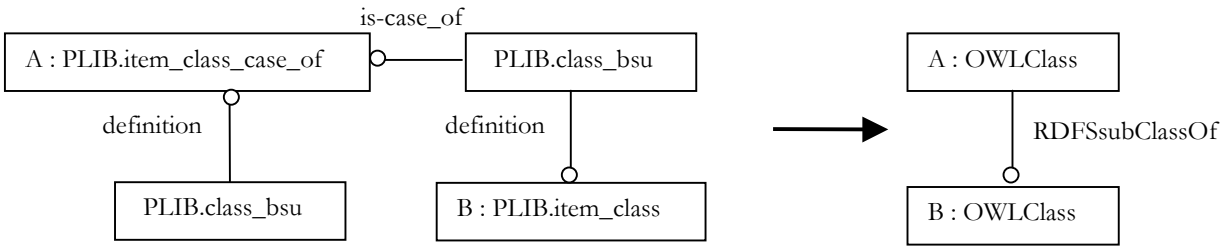


Figure 6.5 : Mise en correspondance d'une relation d'importation PLIB

Remarque : PLIB.item_class_case_of est une sous-classe de PLIB.item_class dont la transformation a été discutée à la section VI.2.1.1.

VI.2.4 Représentation des contraintes sur une classe

Ces contraintes peuvent porter soit sur une propriété, soit sur un type de données.

VI.2.4.1 Cas d'une contrainte portant sur une propriété

D'une manière générale, la traduction en OWL se fera en définissant la classe au niveau de laquelle est définie la contrainte comme étant sous-classe d'une classe anonyme implémentant cette contrainte.

Nous distinguons deux principaux cas :

a) La classe redéfinit le co-domaine d'une propriété

Deux principales situations peuvent se présenter.

1. Le co-domaine est restreint à une classe de l'ontologie

La super -classe anonyme à considérer dans ce cas est celle des individus dont les valeurs de la propriété redefinie sont des instances du co-domaine spécifié.

Illustration

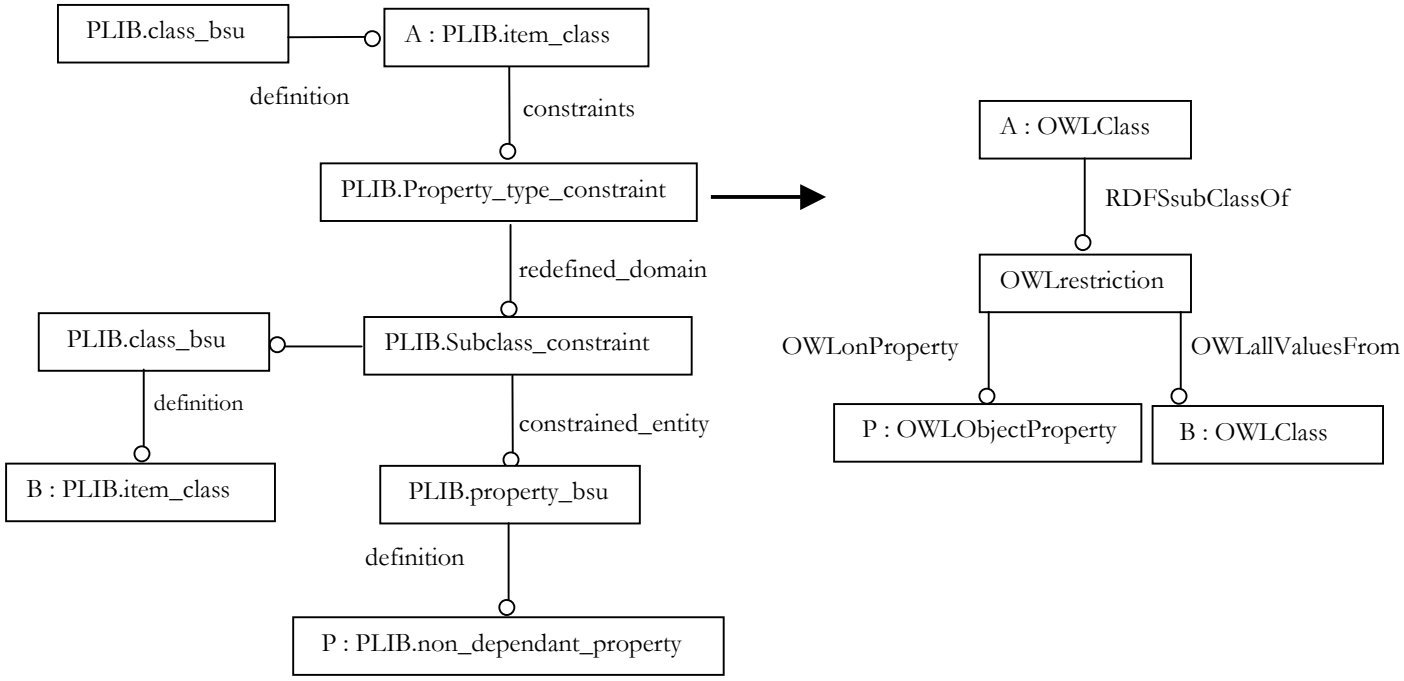


Figure 6.6 : Mise en correspondance d'une contrainte PLIB : restriction du co-domaine à une classe

2. Le co-domaine est restreint à un ensemble énuméré de valeurs possibles

La super-classe anonyme à considérer dans ce cas est celle des individus dont les valeurs de la propriété redéfinie appartiennent à la liste des valeurs spécifiées.

Illustration

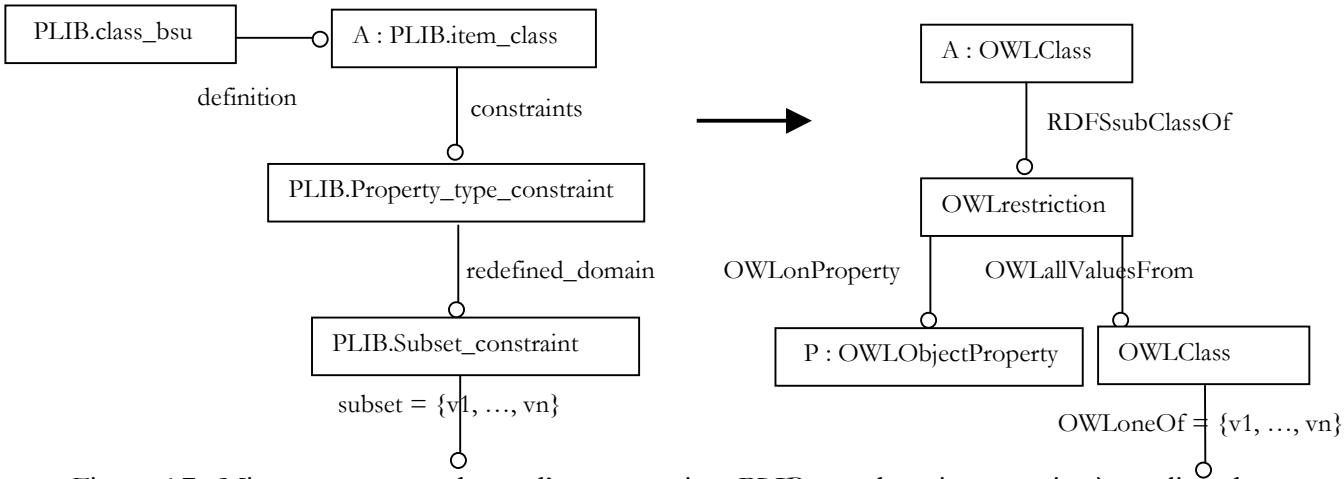


Figure 6.7 : Mise en correspondance d'une contrainte PLIB : co-domaine restreint à une liste de valeurs

b) Autres cas

Pour les contraintes sur les propriétés dépendantes du contexte, les contraintes portant sur la restriction du co-domaine à une expression régulière ou à un intervalle ordonné et les contraintes portant sur la longueur de la chaîne dans le cas d'un propriété de type chaîne de caractères, OWL ne fournit pas de mécanisme pour prendre en compte ces contraintes. Certaines études en cours visant à étendre OWL [Pan & Horrocks 04] pourraient apporter des solutions à ce gendre de constructions comme nous le verrons dans la suite (cf. section VI.3).

VI.2.4.2 Cas d'une contrainte portant sur un type de données

Nous ne pouvons avec OWL spécifier des contraintes sur un type de donnée localement à une classe.

VI.2.5 Représentation des propriétés

Nous distinguons ici deux principaux cas de figures.

VI.2.5.1 Cas d'une propriété essentielle ou d'une propriété non essentielle

D'une manière générale, une propriété PLIB sera convertie en une propriété OWL. Comme pour les classes, nous proposons d'adjoindre un attribut RDFScomment initialisé à « REPRESENTATION_P_DET » afin de distinguer ces deux catégories de propriétés.

Illustration

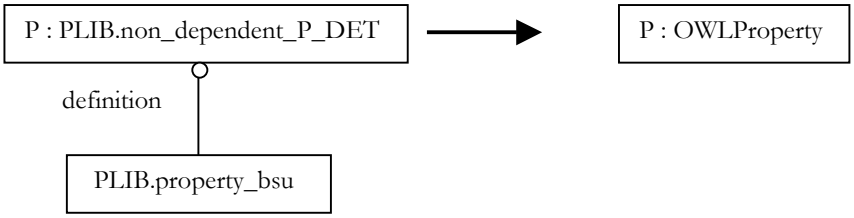


Figure 6.8 : Mise en correspondance d'une propriété PLIB

De même si la propriété est marquée comme désapprouvée, cette information sera prise en compte par l'attribut RDFStype.

Illustration

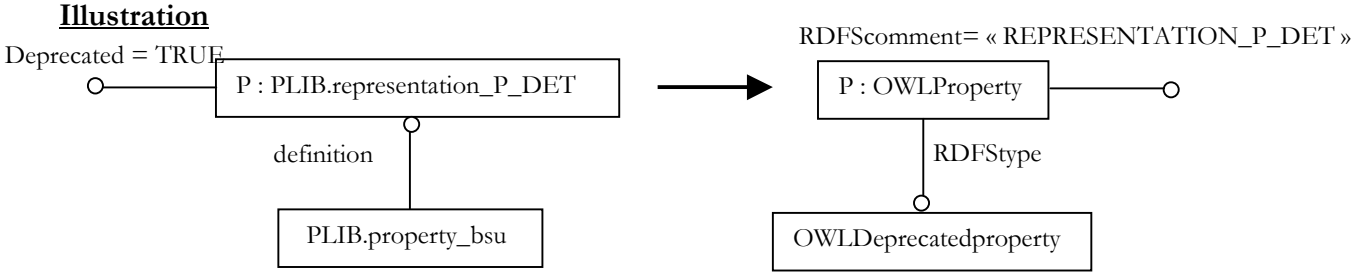


Figure 6.9 : Mise en correspondance d'une propriété PLIB désapprouvée

VI.2.5.2 Cas d'une propriété dépendante du contexte et de ses paramètres de contexte

Ce cas sera traité à la section VI.3 car la solution que nous proposons nécessite une extension du langage OWL.

VI.2.6 Représentation du co-domaine d'une propriété

Deux situations peuvent se présenter

VI.2.6.1 Le co-domaine de la propriété est une classe de l'ontologie

Ce type de propriété sera converti en OWL en une OWLObjectProperty qui un sous-type de OWLProperty dénotant des propriétés dont le co-domaine est une classe de l'ontologie. Son domaine sera la classe correspondant à son domaine de visibilité.

Illustration

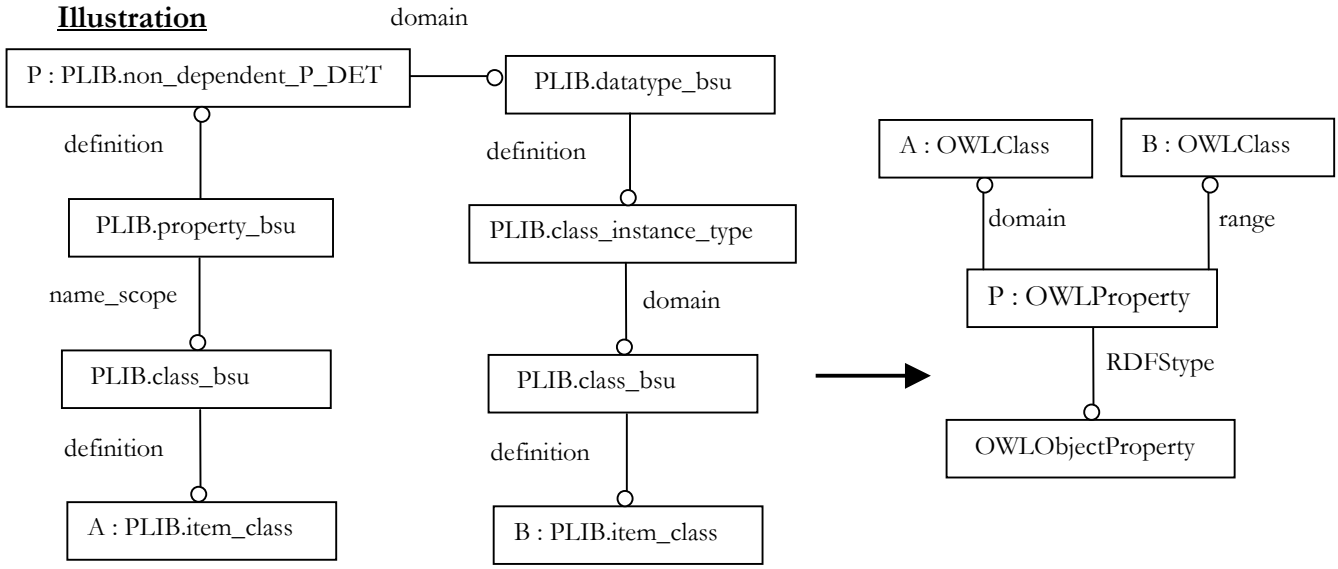


Figure 6.10 : Mise en correspondance d'une propriété objet PLIB

VI.2.6.2 Le co-domaine de la propriété est un type simple PLIB

Ce type de propriété sera converti en OWL en une OWLDatatypeProperty qui un sous-type de OWLProperty dénotant des propriétés dont le co-domaine est un type de données simple (cf. section VI.2.7 pour la correspondance entre les types de données simples PLIB et OWL). Son domaine sera la classe correspondant à son domaine de visibilité.

Illustration

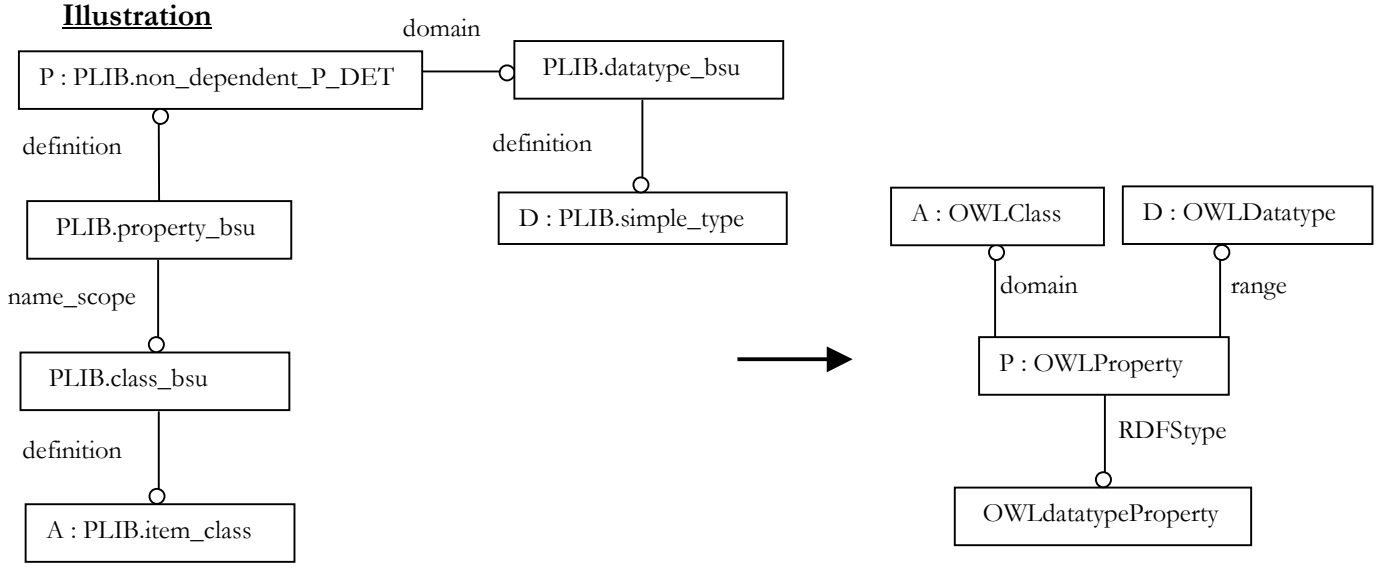


Figure 6.11 : Mise en correspondance d'une propriété simple PLIB

En addition aux deux précédents cas de figures,

a) Si la propriété ne peut avoir au plus une valeur (est une propriété optionelle),

Lors de sa conversion, un attribut supplémentaire RDFStype de type OWLFunctionalProperty sera ajouté.

b) Si la propriété est de type collection ie : peut avoir plus d'une valeur

Nous distinguons deux principaux cas de figures.

1. Son arité minimale est 0 et son arité maximale n'est pas spécifiée

Cette situation est implicitement prise en compte par OWL qui considère que par défaut (si pas de spécification OWLFunctionalProperty) qu'une propriété admet un nombre indéterminé de valeurs.

2. Son arité minimale est égale à n (n >0) (respectivement, et / ou son arité maximale est égale à m (m < ?)

Ce cas sera traduit en OWL en définissant le domaine de visibilité de la propriété comme héritant de la super-classe anonyme des individus ayant au moins n valeurs (respectivement au plus m valurs) pour la propriété.

Illustration

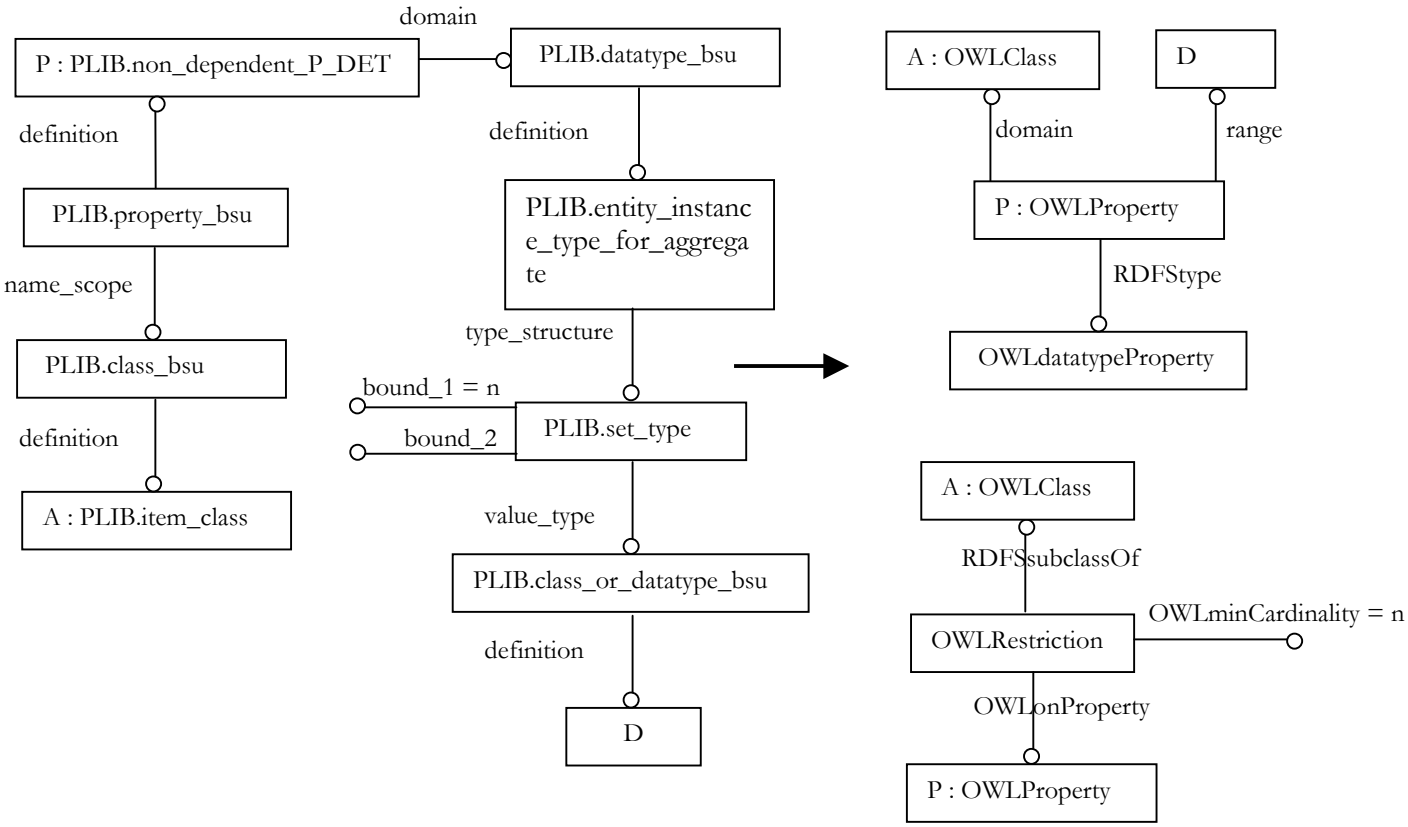


Figure 6.12 : Mise en correspondance d'une propriété PLIB de type collection

Remarque : se référer aux deux cas précédents pour la conversion du co-domaine D

Cette figure s'applique également si bound_2 = m ou bound_1 = bound_2 = k. il suffit pour cela de remplacer OWLminCardinality = n respectivement par : OWLmaxCardinality =m et OWLcardinality = k.

VI.2.7 Représentation des types de données

Le modèle PLIB est muni d'un système de types complexe, qui permet entre autre d'associer des unités à des certains types de bases; ce système est donc très important surtout dans le cadre de l'interopérabilité entre applications.

VI.2.7.1 Correspondance entre types simples PLIB et OWL

Ces correspondances sont données par le tableau ci_dessous.

PLIB	OWL
boolean_type	xsd:boolean
int_type	xsd:decimal
real_type	xsd:double , xsd:float
remote_http_address	xsd:anyURI,
String_type	xsd:string

Tableau 6.1 : Correspondance entre types simples PLIB et OWL

VI.2.7.2 Représentation des types énumérés

Tout type énuméré PLIB sera représenté en un type énuméré OWL.

Illustration

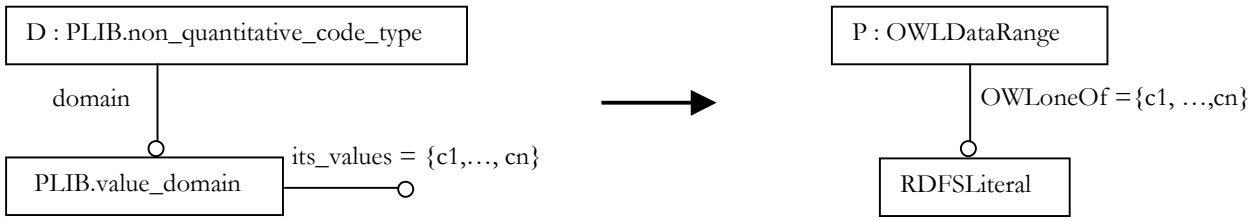


Figure 6.13 : Mise en correspondance d'un type énuméré PLIB

VI.2.7.3 Représentation des types nommés

OWL ne possède pas encore de mécanisme permettant de définir des types nommées à partir des types de base. Nous proposons de traiter un type nommé comme le type de base dont il dérive. Ainsi, le type PLIB.text par exemple sera traité comme un PLIB_string_type.

VI.2.7.4 Représentation des unités de mesures et des unités monétaires

Ils seront traités de la même manière que les types nommés, c'est-à-dire comme les types de base dont ils sont dérivés. Ainsi, les types PLIB tels que int_currency_type, int_measure_type et real_currency_type et real_measure_type seront traités respectivement comme des int_type ou des real_type.

Notons cependant que des travaux dans le cadre de l'interopérabilité des ontologies [Costello 03] sont en cours pour doter OWL de mécanismes pour définir les équivalences via certaines transformations entre des types de données (par exemple : 1 kilomètre = 1,609344 miles).

VI.3 Extensions envisagées d'OWL

Les extensions d'OWL que nous proposons dans cette section visent à permettre :

- la représentation des classes de la catégorie PLIB.functional_view_class.

- la représentation des équivalences fonctionnelles entre les valeurs de propriétés (propriétés dérivées PLIB) : cette équivalence s'exprime comme suit : la propriété P1 est équivalente via une certaine transformation à la propriété P2 ; par exemple : diamètre = rayon * 2.

- la représentation des propriétés dépendantes du contexte et de leurs paramètres de contexte.

Ces propositions se basent sur les travaux de Pan et Horrocks (cf. section III.3.7).

La figure ci-dessous présente une vue simplifiée d'une partie de leur proposition.

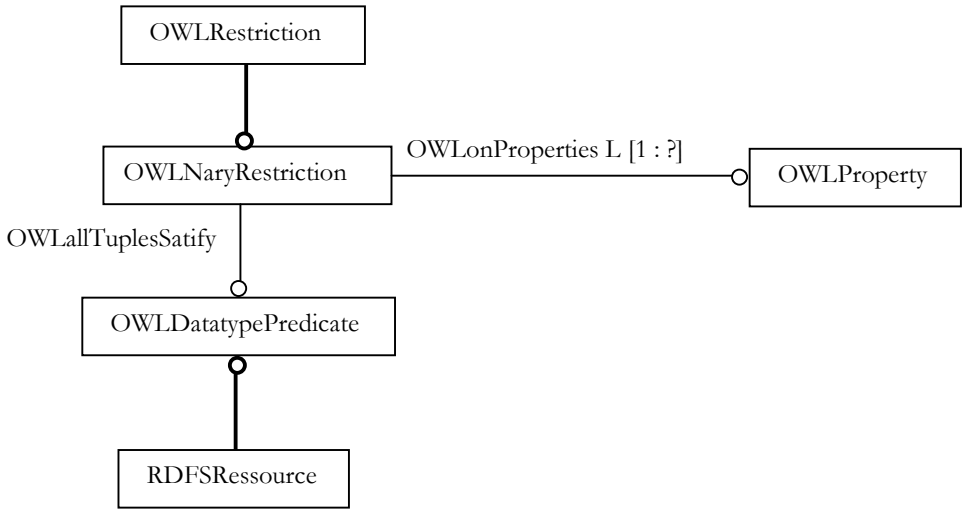


Figure 6.14 : Schéma EXPRESS-G simplifié de la proposition d'extension d'OWL

Dans cette figure, l'entité :

- OWLNaryRestriction est une restriction qui peut porter sur plusieurs propriétés, et,
- OWLDatatypePredicate est une ressource (généralement une fonction booléenne) qui définit la contrainte à respecter par les propriétés sur lesquelles s'applique la restriction.

Remarque : l'ordre d'apparition des propriétés est important ici car, les propriétés doivent correspondre en genre et en nombre avec les arguments de la fonction définissant la contrainte à respecter.

On peut par exemple grâce à leur proposition exprimer la relation reliant les propriétés « diamètre » et « rayon » d'une classe « cercle » en définissant la classe « cercle » comme sous-classe de la classe anonyme des individus satisfaisant le prédicat : diamètre = « rayon * 2.

Illustration

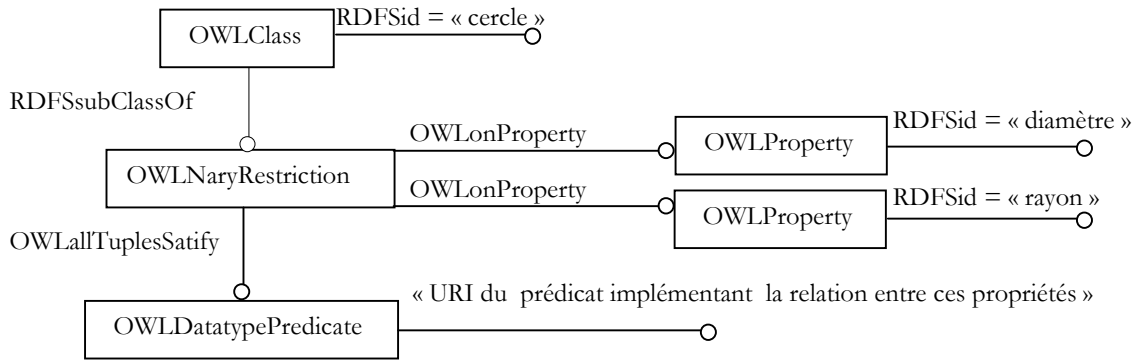


Figure 6.15 : Représentation d'une propriété dérivée avec OWL

Nous pouvons donc d'une manière similaire spécifier :
1. la relation entre toute propriété dérivée et les propriétés dont elle dérive,
2. la relation entre une propriété dépendante du contexte et ses paramètres de contexte,
3. les contraintes portant sur la restriction du co-domaine à une expression régulière ou à un intervalle ordonné et les contraintes portant sur la longueur de la chaîne dans le cas d'une propriété de type chaîne de caractères.

Il est évident que l'idéal dans les deux premiers cas serait non pas d'avoir une fonction booléenne qui spécifie la contrainte que doivent vérifier les propriétés, mais tout simplement une fonction retournant la valeur de propriété dérivée ou celle de la propriété dépendante en fonction de celle de ses paramètres de contexte.

VI.4 Spécification du mapping d'une ontologie PLIB vers OWL

VI.4.1 Règles de transformation retenues

La section précédente (VI.1) nous a permis de mieux analyser les différentes alternatives que nous pouvons envisager pour mettre en correspondance une ontologie PLIB vers OWL, nous allons dans cette section spécifier les règles de transformation que nous avons retenues.

- Toute classe de définition ou de représentation sera transformée en une classe primitive OWL ;
- Il sera automatiquement défini pour la transformée d'une classe de représentation la propriété d'annotation RDFScomment initialisée à chaîne « FUNCTIONAL_MODEL_CLASS » ;
- La relation d'importation sera transformée en la relation d'héritage OWL ;
- Toute classe PLIB redéfinissant le co-domaine d'une propriété sera transformée en une classe OWL sous-classe de la classe anonyme des individus donc les valeurs de la propriété redéfinie sont celles du co-domaine spécifié ;
- Toute propriété essentielle ou non essentielle sera transformée en une propriété OWL ;

- Il sera automatiquement défini pour la transformée d'une propriété non essentielle la propriété d'annotation RDFScomment initialisée à la chaîne « REPRESENTATION_P_DET» ;
- La transformée de toute propriété PLIB ayant pour co-domaine une classe de l'ontologie définira une propriété RDFStype initialisée à OWLObjectProperty ;
- La transformée de toute propriété PLIB ayant pour co-domaine un type simple PLIB définira une propriété RDFStype initialisée à OWLDatatypeProperty ;
- La transformée de toute propriété PLIB optionnelle ou ne pouvant avoir plus d'une valeur définira une propriété RDFStype initialisée à OWLFunctionalProperty ;
- La transformée de toute classe PLIB domaine de visibilité d'une propriété de type collection (pouvant avoir plus d'une valeur) et donc l'arité minimale (n) est supérieure à zéro et / ou l'arité maximale (m) est connue sera sous-classe de la classe anonyme des individus ayant au moins n valeurs et / ou au plus m valeurs pour cette propriété ;
- Tout type défini PLIB sera transformé comme son type de base. ;
- Tout type PLIB représentant une unité de mesure ou une unité monétaire sera transformé comme son type de base.

Les règles que nous venons de retenir peuvent être combinées pour une entité donnée.

VI.4.2 Règles de transformation des propriétés d'annotations

PLIB possède un grand nombre de méta-descripteurs qui servent à décrire très précisément les concepts d'une ontologie. Toutes ces propriétés seront misent en correspondance sur les deux principaux méta-descripteurs d'OWL : RDFSLabel et RDFScomment. Il est donc nécessaire de savoir dans quel(s)attribut(s) se trouvent les valeurs d'une propriété et comment distinguer deux propriétés données. Afin de garder une trace des transformations effectuées dans l'optique par exemple d'une reconversion de OWL vers PLIB, nous proposons de préfixer la valeur de chaque propriété par une chaîne en majuscule qui va nous renseigner sur sa provenance.
Nous proposons la mise en correspondance suivante pour les propriétés d'annotations PLIB :
Chaque valeur de :

- preferred_name($lang_i$, val_j), sera transformée en la propriété RDFSlabel($lang_i$, « PREFERRED_NAME »+val_j);
- short_name($lang_i$, val_j), sera transformée en la propriété RDFSlabel($lang_i$, « SHORT_NAME »+val_j);
- synonymous_name($lang_i$, val_j), sera transformée en la propriété RDFSlabel($lang_i$, « SYNONYMOUS_NAME »+val_j);
- definition($lang_i$, val_j),sera transformée en la propriété RDFScomment($lang_i$, « DEFINITION »+val_j);
- note($lang_i$, val_j), sera mapper sur la propriété RDFScomment($lang_i$, « NOTE »+val_j);
- remark($lang_i$, val_j), sera transformée en la propriété RDFScomment($lang_i$, « REMARK »+val_j);
- revision($lang_i$, val_j),sera transformée en la propriété RDFScomment($lang_i$, « REVISION »+val_j);
- version($lang_i$, val_j), sera transformée en la propriété RDFScomment($lang_i$, « NOTE »+val_j);

- date_of_definition(va1), sera transformée en la propriété RDFScomment(« DATE_OF_DEFINITION »+val);
- date_of_current_version(va1), sera transformée en la propriété RDFScomment(« DATE_OF_CURRENT_VERSION »+val);
- date_of_current_revision(val), sera transformée en la propriété RDFScomment(«DATE_OF_CURRENT_REVISION »+val);

VI.4.3 Règles de transformation des propriétés liées par une fonction de dérivation

Les valeurs de certaines propriétés PLIB, dites propriétés dérivées, se calculent en fonction d'autres propriétés. Dans le cas ou la fonction de calcul est inversible, nous proposons de ne mapper que la propriété à partir de laquelle un grand nombre de propriétés pourraient être déduites. Par exemple, la valeur de la propriété dérivée *absolute_id* d'une classe PLIB s'obtient en appliquant une fonction de calcul inversible sur les propriétés *code*, *version* et *supplier_bsu* ; dans ce cas seule propriété *absolute_id* sera donc mise en correspondance.

VI.5 Mise en œuvre de la correspondance d'une ontologie PLIB vers OWL

Le but de cette section est de formaliser les règles de transformation de PLIB vers OWL.
Comme nous l'avons mentionné au chapitre IV, ces règles de transformations vont nous permettre de réconcilier les différences structurelles entre le modèle PLIB et le schéma EXPRESS du modèle OWL. Il sera calculé une vue dont la déclaration définira la structure d'un fichier OWL sur le schéma EXPRESS d'OWL afin d'obtenir le fichier OWL correspondant à une ontologie source PLIB donnée.

VI.6 Transposition de modèle : le langage EXPRESS-X

EXPRESS-X est un complément déclaratif du langage EXPRESS dont l'objectif est de permettre de créer des représentations alternatives (vues) de modèles express et de spécifier de manière explicite des relations de correspondance (mapping) entre entités de différents schémas express. C'est un langage structuré de mapping de données. Il consiste en des éléments qui permettent une spécification déclarative et non ambiguë des relations entre schémas EXPRESS.
Pour cela, EXPRESS-X supporte deux types de constructeurs spécifiques : SCHEMA_VIEW et SCHEMA_MAP. Respectivement pour des schémas des vues des données d'un modèle EXPRESS ou des schémas de mapping des données de plusieurs schémas EXPRESS.

Un schéma de mapping EXPRESS-X spécifie la correspondance entre les définitions des entités de plusieurs schémas express. Ainsi, la définition d'un type « MAP » spécifie comment les données décrites par un schéma EXPRESS (modèle source) seront transformées en données d'un autre schéma EXPRESS (modèle cible). Un schéma de mapping EXPRESS-X décrit les relations entre les instances d'entités du modèle source et instances d'entités du modèle cible. A partir des critères de

sélection définis, la déclaration du mapping identifie les instances du schéma cible qui seront créées après évaluation du mapping. Le schéma de mapping est exécuté afin de créer des instances du schéma cible.
Dans la figure ci-dessous, nous déclarons un mapping pour faire migrer les instances du schéma *Universitaire* vers le schéma *Personne*.

Schéma source	**Schéma cible**	**Schéma de mapping**
SCHEMA Universitaire ; ENTITY Personne ; ABSTRACT SUPERTYPE OF (ONEOF (Etudiant, Employé)) noSS : NUMBER ; nom : STRING ; prenom : STRING ; age : NUMBER ; DERIVE nom_premon := nom + ‘‘ +prenom ; UNIQUE ur1 : noSS ; END_ENTITY ; ENTITY Etudiant ; SUBTYPE OF(Personne) sa_classe : STRING ; ses_notes : LIST [0 : ?] OF REAL ; END_ENTITY ; ENTITY Employé ; SUBTYPE OF(Personne) salaire : REAL ; END_ENTITY ; ENTITY Etudiant_Salarié ; SUBTYPE OF(Etudiant, Salarié) ; END_ENTITY ; END_SCHEMA	SCHEMA Personne; ENTITY Humain ; id : NUMBER ; nom : STRING ; UNIQUE ur1 : Id ; END_ENTITY ; END_SCHEMA	SCHEMA_MAP Exemple_de_mapping ; REFERENCE FROM Universitaire AS SOURCE ; REFERENCE FROM Person AS TARGET MAP SE_to Personne_map AS hum : Humain ; PARTITION etudiant FROM et u: Universitaire.Etudiant ; SELECT hum.id :=etu.noSS ; hum.nom := etu..nom_prenom ; PARTITION Employé FROM emp : UniversitaireEmployé ; SELECT hum.id :=emp.noSS ; hum.nom := emp.nom_prenom ; END_MAP ; END_SCHEMA

Instances de données	
Instances du schéma source	#1 = EMPLOYE(1345, ‘Franck’, ‘KOM’, 23, 3000) ; #2 = EMPLOYE(1345, ‘Patrick’, ‘FANKAM’, 24, 3000) ; # 3 = ETUDIANT(7562 , ‘Chimène’, ‘FANKAM’, 24, ‘Master2’, ()) ,
Instances du schéma cible	#501 = HUMAIN(1345, ‘Franck KOM’) ; #502 = HUMAIN(2468 , ‘Chimène FANKAM’) ; #502 = HUMAIN(7562 , ‘Patrick FANKAM’) ;

Figure 6.16 : exemple de mapping EXPRESS-X

Ce schéma de mapping transforme les instances des entités *Etudiant* et *Employé* en des instances de l'entité *Humain* de la manière suivante :

- le *noSS* correspond à l'attribut *id* de l'entité *Humain* ;

- l'attribut *nom* de l'entité *Humain* est la concaténation des attributs *nom* et *prénom* des entités *Etudiant* et *Employé*.

Nous voyons sur la figure 6.16 l'application du mapping sur trois instances du schéma source.

La mise en œuvre de la mise en correspondance d'une ontologie PLIB vers une ontologie OWL va se faire en implémentant un schéma de mise en correspondance du modèle PLIB vers le schéma EXPRESS du modèle OWL. Ce schéma est en cours d'implémentation ;

Conclusion

Dans cette partie, Nous avons présenté et analysé les différentes règles de mise en correspondance entre les modèles d'ontologie PLIB et OWL. Nous avons identifié au niveau de chaque modèle, les relations à transformer et défini lorsque cela était possible les règles de mise correspondance de ces relations en relevant à chaque fois, les éventuelles modifications à apporter au modèle cible (PLIB ou OWL) afin de conserver la sémantique des relations et des entités transformés, et en précisant aussi ce que n'était pas conservé lors de la transformation.

Nous avons également présenté dans cette dans cette partie les outils de développement qui sont utilisés dans la mise en œuvre des règles établies.

CONCLUSION GENERALE

CONCLUSION GENERALE

Notre travail consistait à étudier les différentes approches de modélisation à base ontologique et à concevoir des règles de correspondances en vue de permettre leur intégration. La nécessité de l'intégration résultant du constat qu'il est utopiste de penser aboutir à la définition d'une ontologie unique.

Nous nous sommes basée pour notre étude sur les modèles PLIB et OWL qui sont des modèles standard respectivement parmi les approches de modélisation issues des bases de données et celles basées de la logique de Description.

Nous avons tout d'abord étudié les ontologies et plus particulièrement les modèles d'ontologies PLIB et OWL. L'étude que nous avons faite de ces deux modèles a relevé que leur intégration devrait tenir compte non seulement d'une réconciliation structurelle mais aussi de la sémantique de ces modèles ie, des mécanismes offerts par les approches de modélisation sous-jacentes.

Nous avons été amenée dans un premier temps à proposer un schéma EXPRESS-G de OWL afin de représenter PLIB et OWL par un même formalisme de modélisation. Ensuite, nous avons établi différentes possibilités de mise en correspondances des constructeurs et relation de OWL et PLIB et inversement.

Lors de la mise en correspondance du modèle OWL dans le modèle PLIB et inversement, nous avons été confronté à divers problèmes en relation principalement avec la sémantique des axiomes et des constructeurs à convertir. Pour certaines relations, la conversion :

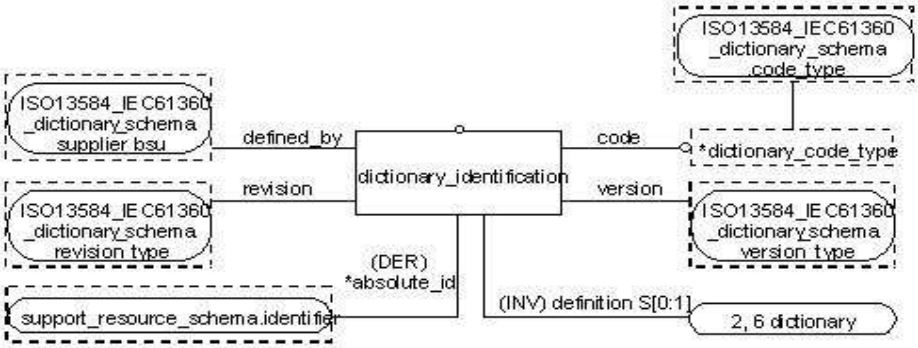
- ne conserve pas toute la sémantique de la relation ;
- n'est pas toujours inversible ;
- n'est pas toujours déterministe ;
- nécessite la création de nouvelles ressources ;
- n'est pas possible.

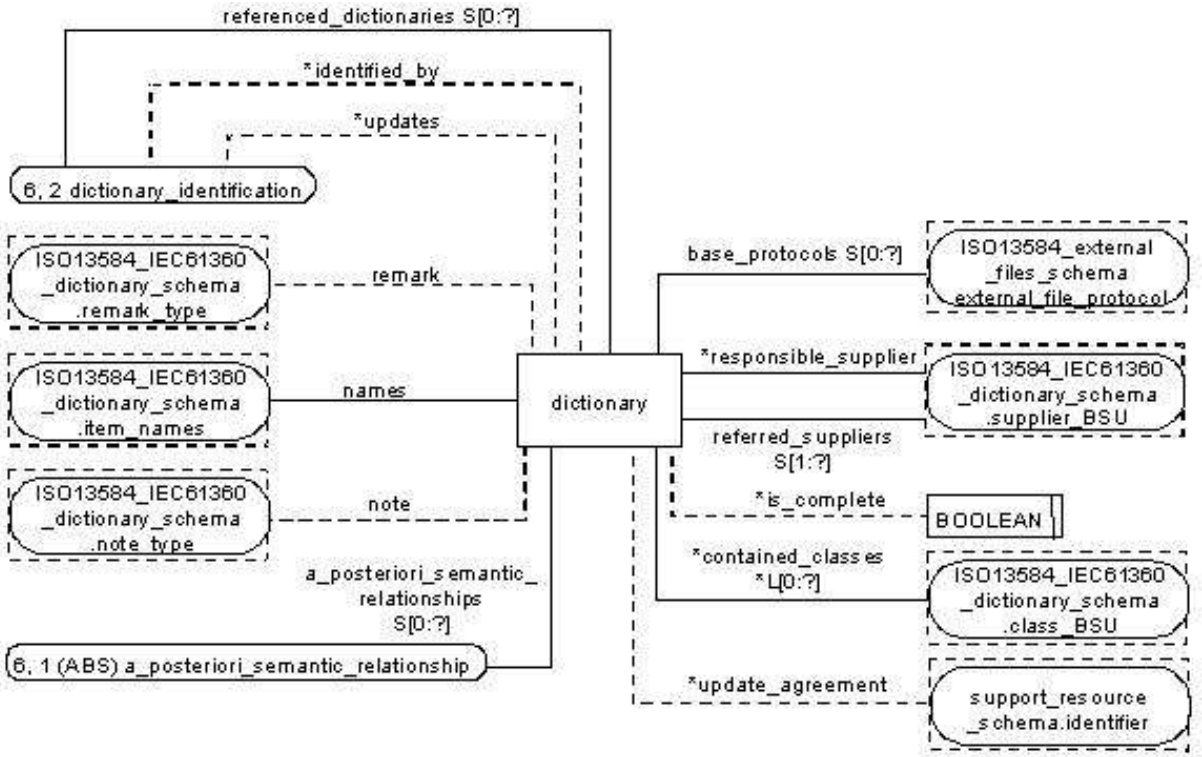
Pour le dernier cas, le modèle cible considéré (PLIB ou OWL) ne permet pas d'exprimer totalement ou même partiellement la sémantique portée par la relation ou le mécanisme à convertir ce qui nous a amené à suggérer des extensions afin qu'ils puissent supporter / modéliser les relations ou les mécanismes à convertir.

L'intégration des approches de modélisation à base ontologique n'inclut pas seulement la réconciliation structurelle des modèles d'ontologie de chaque approche mais aussi l'intégration au sein de chaque approche des mécanismes supportés par les autres approches. L'intérêt de la mise en correspondance de PLIB et OWL est d'inclure / de supporter par OWL les mécanismes de PLIB et inversement d'inclure / de supporter par PLIB les mécanismes de OWL, ce qui va permettre de doter OWL et le Web sémantique d'un support de base de données robuste (bonnes performances de stockage et expression de contraintes d'intégrité), et, les ontologies PLIB de mécanismes d'inférence et de raisonnement. Cette mise en correspondance permet également de définir un lien entre ces deux modèles, et de faciliter l'échange d'informations. Ella va également permettre l'exploitation des ontologies PLIB et OWL par des outils communs.

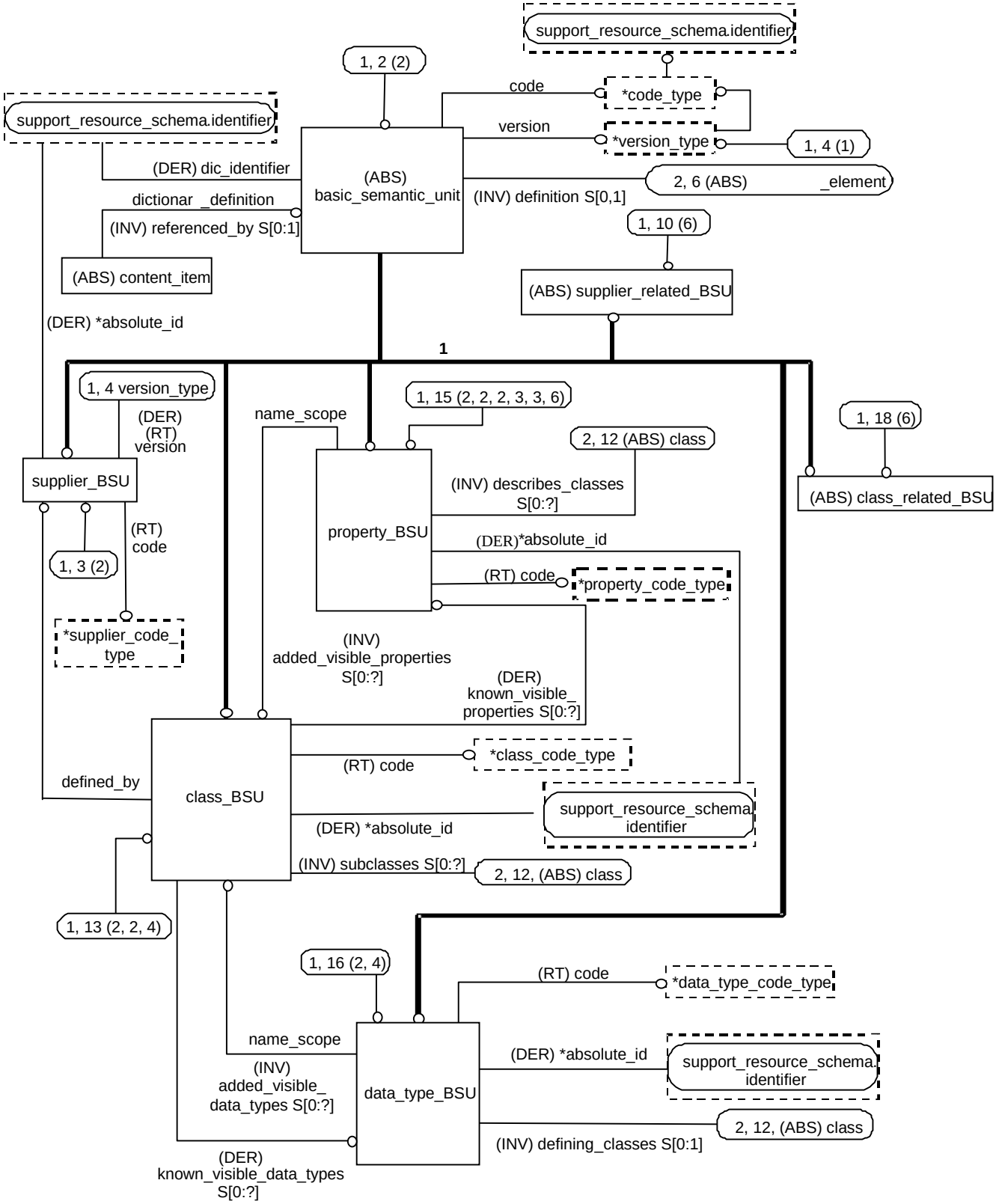
ANNEXES : SCHEMA EXPRESS DU MODELE PLIB

ANNEXE 1 : Le modèle EXPRESS-G de PLIB : dictionnaire

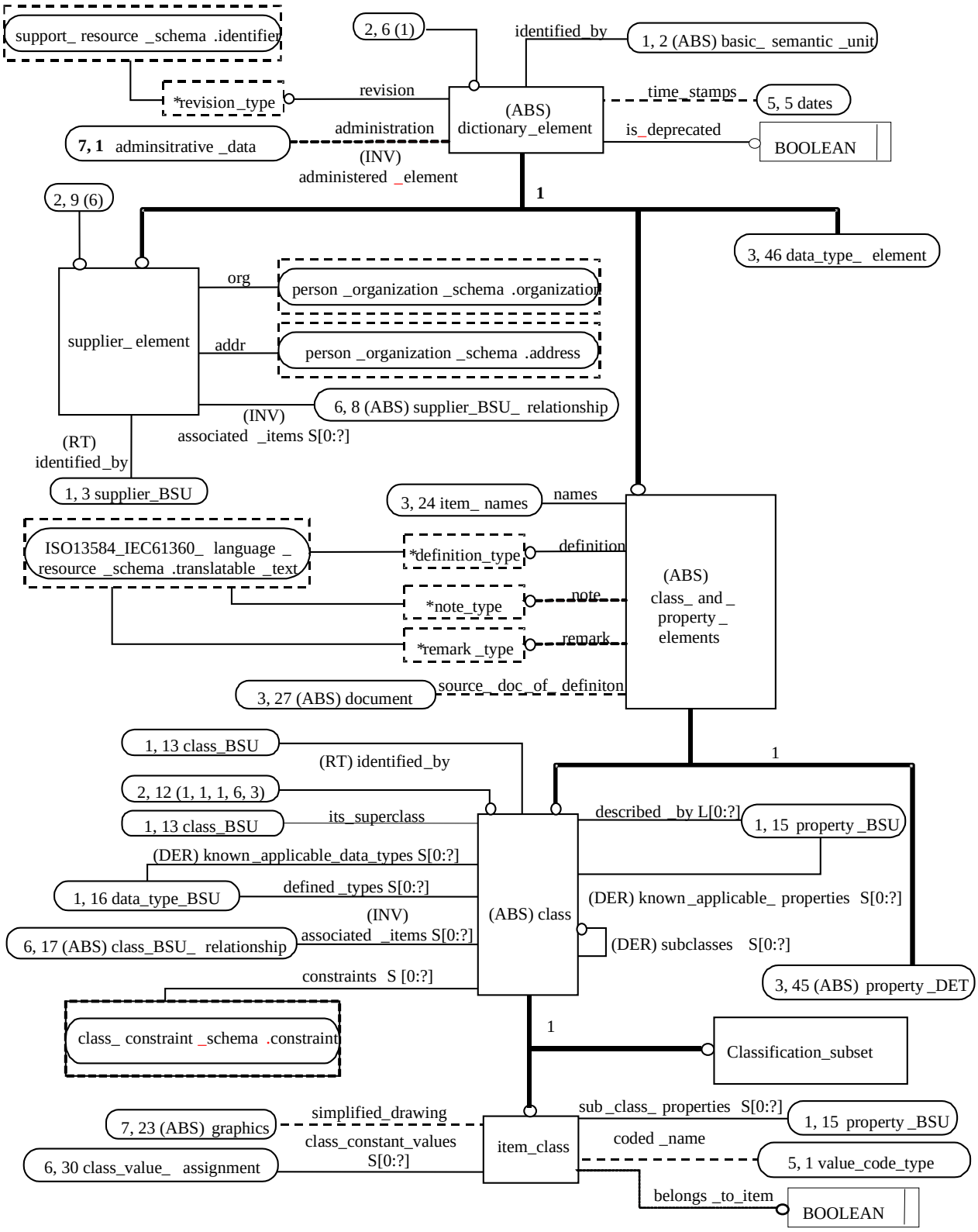




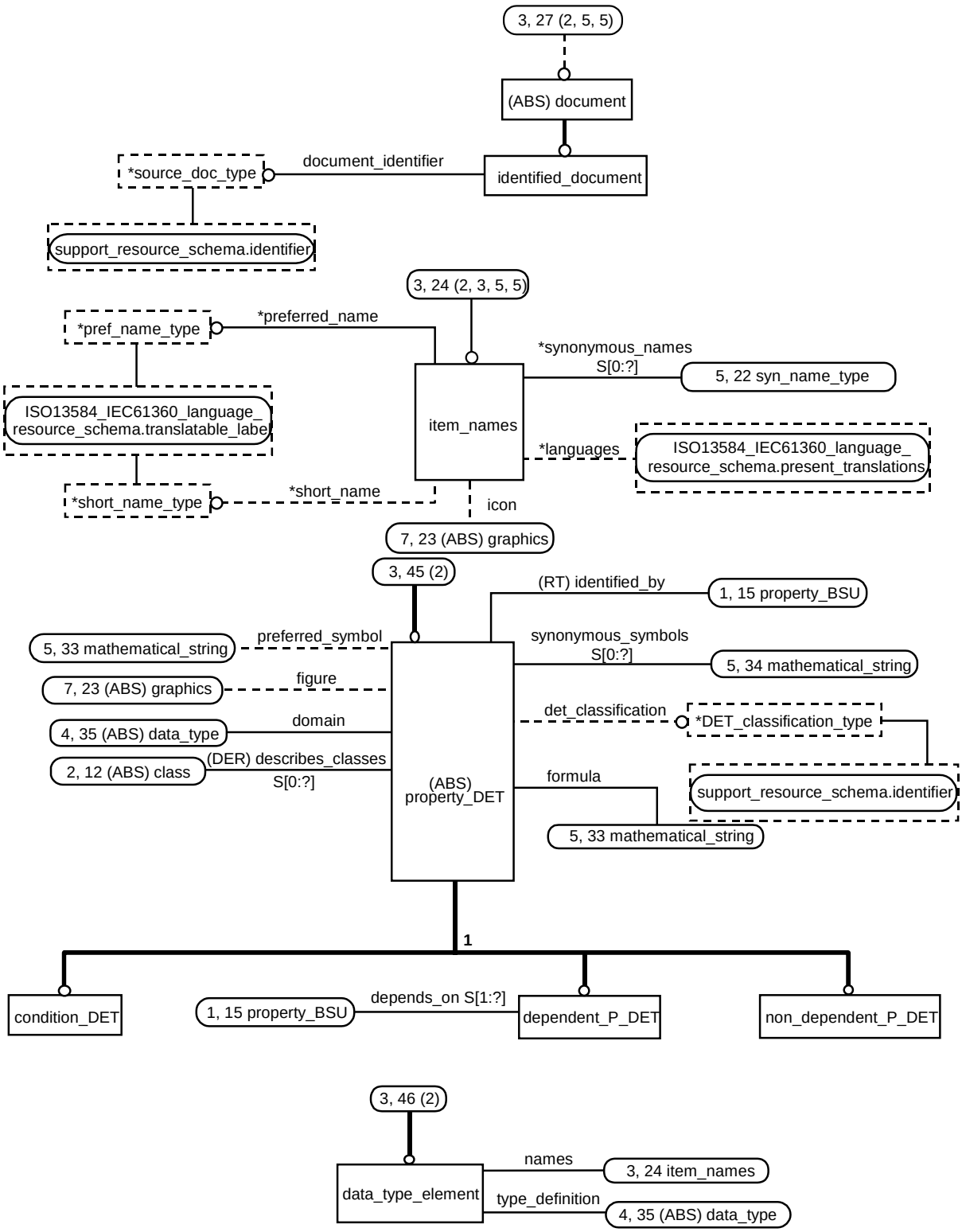
ANNEXE 2 : Le modèle EXPRESS-G de PLIB : bsu



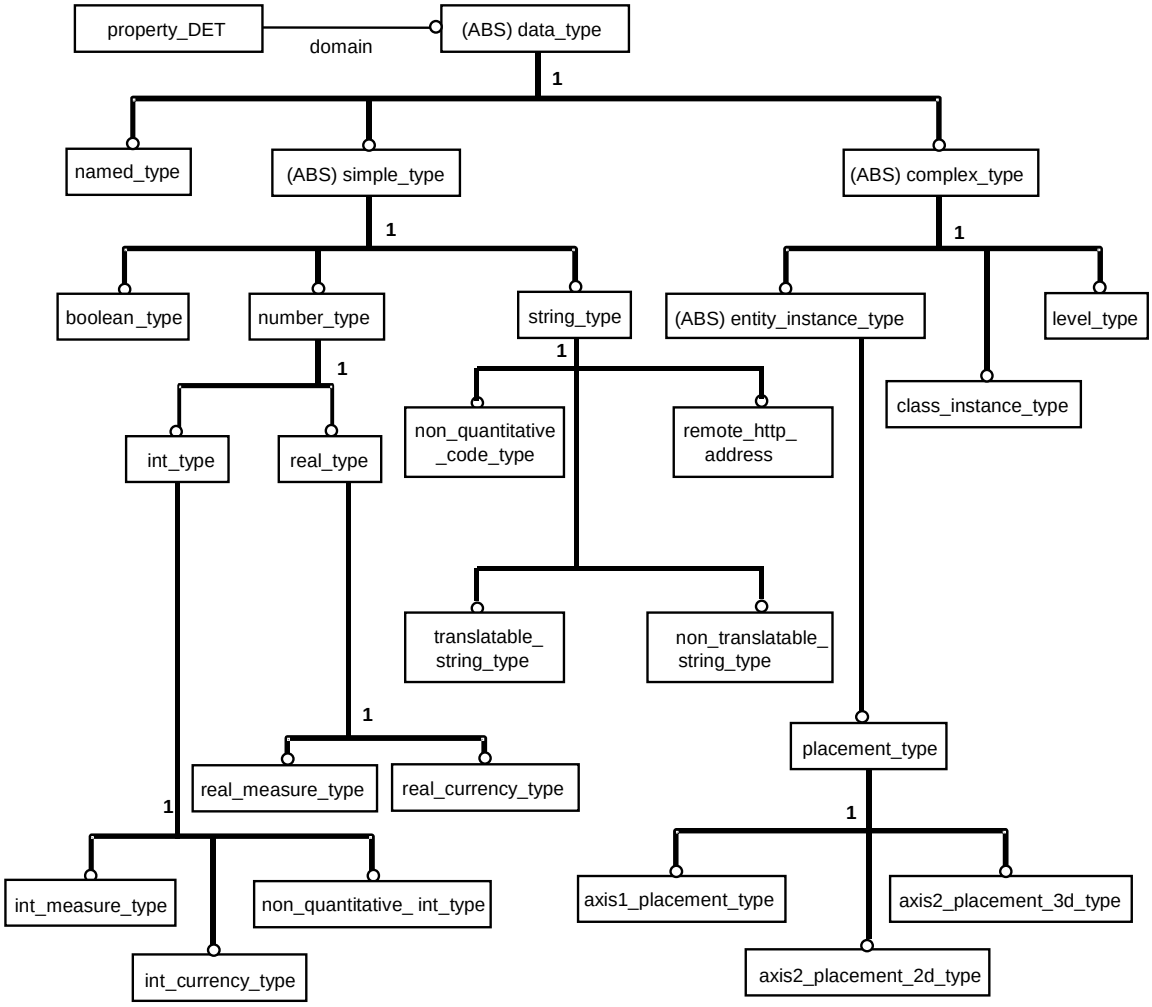
ANNEXE 3 : Le modèle EXPRESS-G de PLIB : classe



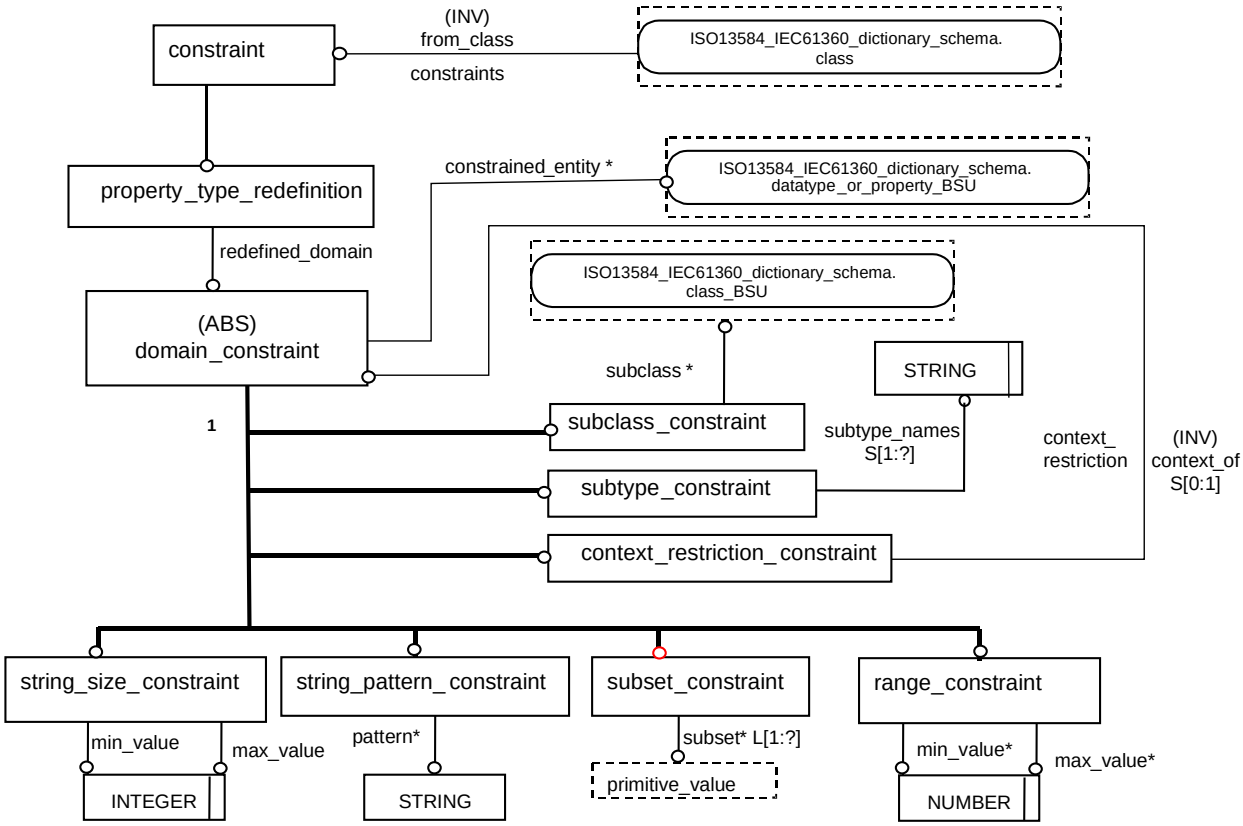
ANNEXE 4 : Le modèle EXPRESS-G de PLIB : propriété



ANNEXE 5 : Le modèle EXPRESS-G de PLIB : types de données



ANNEXE 6 : Le modèle EXPRESS-G de PLIB : contraintes



BIBLIOGRAPHIE

[Ait-Ameur 0] *Ait-ameur Y.*, Representation of procedural knowledge and its use to compute a form of subsumption, Topical day on Semantic Integration of Heterogeneous Data, IFIP World Computer Congress, Toulouse - France, 2004.

[Costello 03] *Costello R. L.,* Enhancing Data Interoperability with Ontologies, Canonical Forms and Include Files, August 10 2003. http://www.xfront.com/interoperability/CanonicalForms.html

[Dieng 06] *Dieng-Kuntz R.,* Ontologies for Knowledge Management, Interoperability Research School, April 13, 2006.

[Gruber 93] *Grube T.*., A translation approach to portable ontology specification, Knowledge Acquisition, 1993, pp. 7.

[Hondjack 02] *Hondjack D.*, Conception d'une architecture de Base de Données à base Ontologique, Mémoire de DEA. Université de Poitiers, Sept 2003, 128 pages, pp. 25-28.

[Hondjack 05] *Hondjack D., Jean S., Nguyen Xuan Dung, Guy Pierra*, Ingénierie dirigée par les modèles en EXPRESS : *un exemple d'application* IDM'05 Premières Journées sur l'Ingénierie Dirigée par les Modèles Paris, 30 juin, 1 juillet 2005. http://idm.imag.fr/idm05/documents/52/P52.pdf

[Bellatreche & al 03] *Bellatreche L., Pierra G., Nguyen Xuan Dung and Hondjack D.*, An Automated Information Integration Technique using an Ontology-based, Appear in : Proceedings of CE'2003, Special track on Data Integration in Engineering, Madeira, Portugal, Juillet 2003. http://www.lisi.ensma.fr/ftp/pub/documents/papers/2003/2003-CE2003-Bellatreche.pdf

[Minski 96] *Minski M.,* The Society of Mind, Edition Simon & Schuster Inc., New York, 1996.

[Napoli 97] *Napoli A.,* Une introduction aux logiques de descriptions, INRIA dec. 97. ftp://ftp.inria.fr/INRIA/publication/publi-pdf/RR/RR-3314.pdf

[Neches 91] *Neches, R., Fikes, R., Finin, T., Gruber, T., Patil, R., Senator, T., and Swartout, W. R.,* Enabling Technology for Knowledge Sharing, AI Magazine, fall1991.

[Pan & Horrocks 04] *Pan J., Horroks I.*, OWL-E: Extending OWL with Expressive Datatype Expressions, 2004. http://dl-web.man.ac.uk/~panz/Zhilin/pubc.php?type=paper&id=PaHo04

[PIE89] *Pierra G.*, Bibliothèque neutre de Composants Standard pour la CAO : le projet européen CAD/LIB, 1989, Revue internationale de CFAO et d'Infographie, vol. 4, n°2, pp. 35-53.

[PIE 94] *Pierra G.*, Représentation et Echange de données techniques, 1994, 19 pages. http://www.lisi.ensma.fr/ftp/pub/documents/papers/2000/2000-MI-Pierra.pdf

[PIE 02a] *Pierra G.*, Developing a new paradigm : Ontology-based information modelling, ISO TC184/SC4/WG2 meeting, Stockolm, Juin 2002. http:// www.ini.dz/conference/ecole2004/presentations-html/pierra.mht

[PIE 02b] *Pierra G.*, Un modèle formel d'ontologie pour l'ingénierie, le commerce électronique et le Web sémantique: Le modèle de dictionnaire sémantique PLIB, Journées Scientifiques WEB SEMANTIQUE, Paris, Octobre, 2002.
http://www.lalic.paris4.sorbonne.fr/stic/octobre/octobre5/16.Pierra.pdf

[PIE 05] *Pierra G.*, *Hondjack D., Jean S., Nguyen Xuan Dung*, Ingénierie dirigée par les modèles en EXPRESS : *un exemple d'application,* IDM'05 Premières Journées sur l'Ingénierie Dirigée par les Modèles Paris, 30 juin, 1 juillet 2005. http://idm.imag.fr/idm05/documents/52/P52.pdf

[RDF 03] RDF Semantics, http://www.w3.org/TR/2003/WD-rdf-mt-20030123/#dtype_interp, W3C working Draft 23 January 2003

[SAR 99] *Sardet E.*, Intégration des approches modélisation conceptuelle et structuration documentaire pour la saisie, la représentation, l'échange et l'exploitation d'informations. Application aux catalogues de composants industriels, Thèse de Doctorat en Informatique. Université de Poitiers et ENSMA, 1999, 199 pages.

[Schreiber 02] *Schreiber G.,* The Web is not Well-Formed. Issues in Developing a Web Ontology Language,
http://www.informatiewetenschap.org/docs/schreiber.ppt, W3C's Web Ontology Working Group, WG Infwet, 7 juni 2002.

[Jean 04] *Jean S*, Langage d'exploitation de base de données ontologiques, Mémoire de DEA. Université de Poitiers, Juin 2004, 93 pages. http://www.lisi.ensma.fr/members/jean/download/rapport.pdf

[Tim Bernes-Lee 99] *Tim Berners-Lee, Mark Fischetti. Harper*, Weaving the Web : The original design and ultimate destiny of the World Wide Web by its Inventors, San Francisco, 1999.

TABLE DES MATIERES

SOMMAIRE ..... **II**

REMERCIEMENTS ..... **IV**

RESUME ..... **V**

ABSTRACT ..... **VI**

INTRODUCTION GENERALE **2**

PARTIE I : PRESENTATION GENERALE **4**

INTRODUCTION **5**
CHAPITRE I : LES ONTOLOGIES EN INGENIERIE DES DONNES. **6**
I.1 LA NOTION D'ONTOLOGIE 6
I.1.1 Définition 6
I.1.2 Structure d'une ontologie 6
I.1.3 Classification des ontologies 7
I.2 QUELQUES DOMAINES D'APPLICATION DES ONTOLOGIES 7
I.2.1 Les bases de données 7
I.2.2 Le WEB sémantique [Tim Berners-LeeO5] 8
I.2.3 L'intégration de sources de données hétérogènes 10
I.2.4 Le traitement du langage naturel 10
I.3 POURQUOI INTÉGRER LES APPROCHES DE MODÉLISATION À BASE D'ONTOLOGIE ? 10
CHAPITRE II : LE MODELE D'ONTOLOGIE PLIB **12**
II.1 LES CARACTÉRISTIQUES D'UNE ONTOLOGIE PLIB 12
II.2 LES CONSTRUCTEURS DU MODÈLE PLIB [PIE 02] 12
II.2.1 Identification des concepts 13
II.2.2 Les constructeurs pour définir les concepts 13
II.2.3 Les constructeurs pour définir les propriétés 13
II.2.4 Le système de type 14
II.2.5 Définition des instances d'un concept 14
II.3 GESTION DU CONTEXTE DANS LE MODÈLE PLIB 15
II.3.1 La modélisation des concepts 15
II.3.2 L'évaluation des concepts 15
II.3.3 Les principes fondamentaux de la modélisation d'ontologies PLIB 16
II.4 MODÈLE FORMEL D'UNE ONTOLOGIE PLIB [PIE 05] 16

II.5 ONTODB : UN MODÈLE D'ARCHITECTURE POUR LES BASES DE DONNÉES À BASE ONTOLOGIQUES.... 17
II.5.1 Présentation 17
II.5.2 Extensions prévues d'OntoDB..... 18
CHAPITRE III : LE LANGAGE D'ONTOLOGIE OWL..... 19
III.1 RDF 19
III.1.1 Caractéristiques de RDF..... 20
III.1.2 Le modèle formel de RDF..... 21
III.1.2.1 Définition d'un schéma RDF..... 22
III.1.3 De RDF à RDF Schéma : les insuffisances de RDF..... 22
III.2 RDFS..... 22
III.2.1 Présentation 22
III.2.2 Caractéristiques de RDF Schema 23
III.2.3 Le modèle formel de RDFS..... 24
III.2.3.1 Définition d'un schéma RDFS 24
III.2.3.2 Définition d'une instance (statement) RDFS 24
III.2.4 De RDF Schema à OWL : les insuffisances de RDF Schéma..... 24
III.3 OWL..... 25
III.3.1 Ambitions de OWL 25
III.3.2 Les influences qui ont guidé la conception d'OWL..... 26
III.3.3 OWL : une variante des logiques de description..... 26
III.3.3.1 Modélisation de la connaissance 26
III.3.3.2 Les opérations à la base du raisonnement terminologique..... 27
III.3.3.3 Les constructeurs OWL issus des logiques de descriptions 28
III.3.3.4 Syntaxe des Logiques de Description..... 28
III.3.4 Identification des ressources 30
III.3.5 Les constructeurs et axiomes OWL..... 30
III.3.5.1 Les constructeurs de classes 31
III.3.5.2 Les constructeurs de propriétés 32
III.3.5.3 Les types de données..... 34
III.3.5.4 Les individus 34
III.3.5.5 Autres concepts 35
III.3.6 Les sous-langages d'OWL 35
III.3.6.1 OWL Full 36
III.3.6.2 OWL DL..... 36
III.3.6.3 OWL lite..... 37
III.3.7 Extensions prévues d'OWL [Pan & Horrocks 04]..... 38
CHAPITRE IV : CONVERSION DE SCHEMA ENTRE MODELES DE DONNEES..... 40
IV.1 CHOIX DU FORMALISME DE MODÉLISATION..... 41
IV.2 LE LANGAGE EXPRESS..... 42
IV.2.1 Introduction [PIE 02] 42
IV.2.1 La représentation graphique EXPRESS-G..... 43
IV.3 LE SCHÉMA EXPRESS-G D'OWL 44
IV.3.1 Schéma EXPRESS-G de RDFS 44
IV.3.2 Schéma EXPRESS-G d'OWL : ontologie 45
IV.3.3 Schéma EXPRESS-G d'OWL : classe..... 45
IV.3.4 Schéma EXPRESS-G d'OWL : restriction..... 46
IV.3.5 Schéma EXPRESS-G d'OWL : propriété..... 47
IV.3.6 Schéma EXPRESS-G d'OWL : instance 47
IV.3.7 Schéma EXPRESS-G d'OWL : types de données (1) 48
IV.3.8 Schéma EXPRESS-G d'OWL : types de données (2) 49
CONCLUSION 50

PARTIE II : ANALYSE ET CONCEPTION DES REGLES DE MAPPING DES MODELES PLIB ET OWL..... 51

INTRODUCTION..... 52
CHAPITRE V : CONVERSION DE OWL VERS PLIB 53
V.1 ANALYSE DE LA DÉMARCHE DE TRANSFORMATION..... 53
V.1.1 Représentation d'une classe 53
V.1.1.1 Déterminer la catégorie de classe PLIB sémantiquement proche..... 53
V.1.1.2 Introduire une nouvelle catégorie de classe dans PLIB..... 54
V.1.2 Cas particulier des classes définies..... 54
V.1.2.1 Classe définie par énumération de ses instances 54
V.1.2.2 Classe définie comme étant le complémentaire d'une autre classe..... 55
V.1.2.3 Classe définie comme étant une intersection..... 55
V.1.2.4 Classe définie comme étant une union ou une restriction 55
V.1.2.5 Extension des classes définies..... 55
V.1.3 Représentation de l'héritage..... 56
V.1.3.1 Classe ayant pour super classe une ou plusieurs classes nommées définies 56
V.1.3.2 Classe ayant pour super classe une ou plusieurs classes primitives 56
V.1.3.2 Classe ayant pour super classe une ou plusieurs classes anonymes 58
V.1.4 Assertions sur les classes..... 61
V.1.4.1 Classe définie par l'axiome <owl : equivalentClass> 61
V.1.5 Représentation d'une propriété..... 61
V.1.5.1 Déterminer la catégorie de propriété PLIB sémantiquement proche..... 61
V.1.5.2 Introduire une nouvelle hiérarchie de propriété dans PLIB..... 62
V.1.6 Prise en compte des catégories de propriétés OWL 63
V.1.6.1 Cas d'une propriété objet..... 63
V.1.6.1 Cas d'une propriété simple..... 63
V.1.7 Représentation des sous propriétés..... 63
V.1.8 Conversion du domaine et du co-domaine d'une propriété..... 64
V.1.8.1 Cas d'une propriété ayant comme domaine ou co-domaine implicite owlThing..... 64
V.1.8.2 Cas d'une propriété ayant un ou plusieurs domaines explicites..... 64
V.1.8.3 Cas d'une propriété ayant un ou plusieurs co-domaines 65
V.1.9 Représentation des caractéristiques des propriétés..... 66
V.1.9.1 Cas d'une propriété fonctionnelle..... 66
V.1.9.2 Cas d'une propriété inverse fonctionnelle..... 66
V.1.9.3 Cas d'une propriété réflexive, symétrique ou transitive..... 66
V.1.9.4 Cas d'une propriété ayant une propriété inverse 67
V.1.10 Prise en compte de l'axiome disjointProperties 68
V.1.11 Conversion des types de données..... 68
V.1.11.1 Cas des types de données primitifs..... 68
V.1.11.2 Cas des types de données énumérées 69
V.1.11.3 Cas des types de données définis par les utilisateurs..... 69
V.2 SPÉCIFICATION DU MAPPING D'UNE ONTOLOGIE OWL VERS LE MODÈLE PLIB..... 70
V.2.1 Principes de transformations 70
V.2.2 Règles de transformation des propriétés d'annotations..... 71
V.2.2 Règles de calcul du domaine et du co-domaine d'une propriété..... 72
V.3 MISE EN ŒUVRE DE LA CORRESPONDANCE D'UNE ONTOLOGIE OWL VERS PLIB..... 72
CHAPITRE VI : CONVERSION DE PLIB VERS OWL..... 74
VI.1 CRITÈRE DE SÉLECTION DES ENTITÉS À CONVERTIR 74
VI.2 ANALYSE DE LA DÉMARCHE DE TRANSFORMATION 74

VI.2.1 Représentation d'une classe 74
VI.2.1.1 Cas d'une classe de définition 75
VI.2.1.2 Cas d'une classe de représentation 75
VI.2.1.2 Cas d'une classe de vue 76
VI.2.2 Représentation de l'héritage 76
VI.2.3 Représentation de l'importation 76
VI.2.4 Représentation des contraintes sur une classe 77
VI.2.4.1 Cas d'une contrainte portant sur une propriété 77
VI.2.4.2 Cas d'une contrainte portant sur un type de données 78
VI.2.5 Représentation des propriétés 78
VI.2.5.1 Cas d'une propriété essentielle ou d'une propriété non essentielle 78
VI.2.5.2 Cas d'une propriété dépendante du contexte et de ses paramètres de contexte 79
VI.2.6.1 Le co-domaine de la propriété est une classe de l'ontologie 79
VI.2.6.2 Le co-domaine de la propriété est un type simple PLIB 80
VI.2.7 Représentation des types de données 81
VI.2.7.1 Correspondance entre types simples PLIB et OWL 82
VI.2.7.2 Représentation des types énumérés 82
VI.2.7.3 Représentation des types nommés 82
VI.2.7.4 Représentation des unités de mesures et des unités monétaires 82
VI.3 EXTENSIONS ENVISAGÉES D'OWL 83
VI.4 SPÉCIFICATION DU MAPPING D'UNE ONTOLOGIE PLIB VERS OWL 84
VI.4.1 Règles de transformation retenues 84
VI.4.2 Règles de transformation des propriétés d'annotations 85
VI.4.3 Règles de transformation des propriétés liées par une fonction de dérivation 86
VI.5 MISE EN ŒUVRE DE LA CORRESPONDANCE D'UNE ONTOLOGIE PLIB VERS OWL 86
VI.6 Transposition de modèle : le langage EXPRESS-X 86
CONCLUSION 89

CONCLUSION GENERALE 91

ANNEXES : SCHEMA EXPRESS DU MODELE PLIB 92

ANNEXE 1 : LE MODÈLE EXPRESS-G DE PLIB : DICTIONNAIRE 93
ANNEXE 2 : LE MODÈLE EXPRESS-G DE PLIB : BSU 94
ANNEXE 3 : LE MODÈLE EXPRESS-G DE PLIB : CLASSE 95
ANNEXE 4 : LE MODÈLE EXPRESS-G DE PLIB : PROPRIÉTÉ 96
ANNEXE 5 : LE MODÈLE EXPRESS-G DE PLIB : TYPES DE DONNÉES 97
ANNEXE 6 : LE MODÈLE EXPRESS-G DE PLIB : CONTRAINTES 98

BIBLIOGRAPHIE 99

TABLE DES MATIERES 101

LISTE DES FIGURES 106

LISTE DES TABLEAUX 107

GLOSSAIRE..... 108

LISTE DES FIGURES

FIGURE 2.1 : LE MODÈLE D'ARCHITECTURE ONTODB 18
FIGURE 3.1 : EXEMPLE DE DÉCLARATION RDF 20
FIGURE 3.2 : EXEMPLE DE GRAPHE RDF 21
FIGURE3.3 : EXEMPLE DE GRAMMAIRE DE LOGIQUE DE DESCRIPTION : LE LANGAGE MINIMAL AL 28
FIGURE 3.4 : LES SOUS-LANGAGES DE OWL 37
FIGURE 4.1 : EXEMPLE DE SCHÉMA EXPRESS-G 43
FIGURE 4.2 : SCHÉMA EXPRESS-G DE RDFS 44
FIGURE 4.3 : SCHÉMA EXPRESS-G D'OWL : ONTOLOGIE. 45
FIGURE 4.4 : SCHÉMA EXPRESS-G D'OWL : CLASSE 45
FIGURE 4.5 : SCHÉMA EXPRESS-G D'OWL : RESTRICTION 46
FIGURE 4.6 : SCHÉMA EXPRESS-G D'OWL : PROPRIÉTÉ 47
FIGURE 4.7 : SCHÉMA EXPRESS-G D'OWL : INSTANCE 47
FIGURE 4.8 : SCHÉMA EXPRESS-G D'OWL : TYPES DE DONNÉES 48
FIGURE 4.8 : SCHÉMA EXPRESS-G D'OWL : TYPES DE DONNÉES 49
FIGURE 5.1A : MISE EN CORRESPONDANCE D'UNE CLASSE OWL. 53
FIGURE 5.1B: MISE EN CORRESPONDANCE D'UNE CLASSE OWL. 54
FIGURE 5.2 : MISE EN CORRESPONDANCE D'UNE CLASSE ÉNUMÉRÉE OWL. 54
FIGURE 5.3 : MISE EN CORRESPONDANCE D'UNE RELATION D'HÉRITAGE SIMPLE OWL. 56
FIGURE 5.4 : MISE EN CORRESPONDANCE D'UNE RELATION D'HÉRITAGE MULTIPLE OWL. 57
FIGURE 5.5 : MISE EN CORRESPONDANCE D'UNE SUPER-CLASSE ANONYME : REDÉFINITION DE L'ARITÉ. 58
FIGURE 5.6 : MISE EN CORRESPONDANCE D'UNE SUPER-CLASSE ANONYME : REDÉFINITION DU CO-DOMAINE. 59
FIGURE 5.7A : MISE EN CORRESPONDANCE D'UNE PROPRIÉTÉ OWL. 61
FIGURE 5.7B : MISE EN CORRESPONDANCE D'UNE PROPRIÉTÉ OWL. 62
FIGURE 5.8 : MISE EN CORRESPONDANCE D'UNE PROPRIÉTÉ OBJET OWL. 63
FIGURE 5.9 : MISE EN CORRESPONDANCE D'UNE PROPRIÉTÉ SIMPLE OWL. 63
FIGURE 5.10 : MISE EN CORRESPONDANCE DU DOMAINE D'UNE PROPRIÉTÉ OWL 65
FIGURE 5.11 : MISE EN CORRESPONDANCE DU CO-DOMAINE D'UNE PROPRIÉTÉ OWL 65
FIGURE 5.12 : MISE EN CORRESPONDANCE D'UNE PROPRIÉTÉ FONCTIONNELLE OWL. 66
FIGURE 5.13 : TRANSFORMATION D'UN TYPE DE DONNÉES ÉNUMÉRÉ OWL 69
FIGURE 5.14 : TRANSFORMATION D'UN TYPE DÉFINIT OWL : CONTRAINTE DE LONGUEUR 69
FIGURE 6.1 : MISE EN CORRESPONDANCE D'UNE CLASSE DE DÉFINITION PLIB 75
FIGURE 6.2 : MISE EN CORRESPONDANCE D'UNE CLASSE DE REPRÉSENTATION PLIB. 75
FIGURE 6.3 : MISE EN CORRESPONDANCE D'UNE CLASSE PLIB DÉSAPPROUVÉE 75
FIGURE 6.4 : MISE EN CORRESPONDANCE D'UNE RELATION D'HÉRITAGE PLIB 76
FIGURE 6.5 : MISE EN CORRESPONDANCE D'UNE RELATION D'IMPORTATION PLIB 76
FIGURE 6.6 : MISE EN CORRESPONDANCE D'UNE CONTRAINTE PLIB : RESTRICTION DU CO-DOMAINE À UNE CLASSE .. 77
FIGURE 6.7 : MISE EN CORRESPONDANCE D'UNE CONTRAINTE PLIB : CO-DOMAINE RESTREINT À UNE LISTE DE VALEURS 78
FIGURE 6.8 : MISE EN CORRESPONDANCE D'UNE PROPRIÉTÉ PLIB 78
FIGURE 6.9 : MISE EN CORRESPONDANCE D'UNE PROPRIÉTÉ PLIB DÉSAPPROUVÉE 79
FIGURE 6.10 : MISE EN CORRESPONDANCE D'UNE PROPRIÉTÉ OBJET PLIB 79
FIGURE 6.11 : MISE EN CORRESPONDANCE D'UNE PROPRIÉTÉ SIMPLE PLIB 80
FIGURE 6.12 : MISE EN CORRESPONDANCE D'UNE PROPRIÉTÉ PLIB DE TYPE COLLECTION 81
FIGURE 6.13 : MISE EN CORRESPONDANCE D'UN TYPE ÉNUMÉRÉ PLIB 82
FIGURE 6.14 : SCHÉMA EXPRESS-G SIMPLIFIÉ DE LA PROPOSITION D'EXTENSION D'OWL 83
FIGURE 6.15 : REPRÉSENTATION D'UNE PROPRIÉTÉ DÉRIVÉE AVEC OWL 84
FIGURE 6.16 : EXEMPLE DE MAPPING EXPRESS-X 87

LISTE DES TABLEAUX

TABLEAU 3.1 : COMPARAISON DES SYNTAXES DE OWL 30
TABLEAU 3.2 : LES AXIOMES DE CLASSES OWL 31
TABLEAU 3.3 : LES EXPRESSIONS DE CLASSES OWL 32
TABLEAU 3.4 : LES AXIOMES DE HIÉRARCHIE DE CLASSE OWL 32
TABLEAU 3.5 : LES AXIOMES DE PROPRIÉTÉ OWL 33
TABLEAU 3.6 : ASSERTIONS SUR LES INDIVIDUS OWL 35
TABLEAU 5.1 : STOCKAGE DES CARACTÉRISTIQUE DES PROPRIÉTÉS OWL 67
TABLEAU 5.2 : CORRESPONDANCE ENTRE TYPES SIMPLES OWL ET PLIB 68
TABLEAU 6.1 : CORRESPONDANCE ENTRE TYPES SIMPLES PLIB ET OWL 82

GLOSSAIRE

Assertion : phrase mathématique, à laquelle il est possible, dans le cadre d'une théorie, d'attribuer une valeur de vérité vraie ou fausse, mais pas les deux.

Attributs : caractéristiques (ou des propriétés) des entités qui ne possèdent pas une existence propre, et qui peut être observés ou mesurés. La valeur d'un attribut appartient à type simple (ou atomique) : entier, chaîne de caractères etc.

Base de données : ensemble organisé et intégré de données permettant l'accès concurrent. Elle correspond à une représentation fidèle de l'information et elle s'abstrait autant que possible de contraintes imposées par le matériel. Elle assure la persistance, la cohérence et la concurrence des accès aux données et peut être utilisée pour n'importe quelle application sans duplication des données.

Base de données à base ontologique (BDBO) : base de données qui présente deux caractéristiques. D'une part, elle permet de gérer à la fois des ontologies et des données. D'autres part, elle permet d'associer à chaque donnée le concept ontologique qui en définit le sens.

Classe : description de la structure, du comportement et des contraintes associées à ses instances

Concept défini : concept dont les conditions nécessaires et suffisantes d'appartenance de ses individus sont fournies.

Concept primitif : concept dont seules les conditions nécessaires d'appartenance de ses individus sont fournies

Entité [Chen] : un objet, un fait ou encore un événement du monde réel qui peut être clairement identifié. En règle générale, une entité correspond à un nom commun du texte décrivant le problème à traiter.

Entité [définition normalisée de l'AFNOR/ISO] : tout objet ou association d'objets, concret ou abstrait, existant, ayant existé ou pouvant exister. Elément actif d'un sous-système.

EXPRESS : langage de modélisation des données conçu dans le cadre du projet STEP (STandard for Exchange of Product Model Data). Son objectif principal est la description de modèles d'informations dans le domaine technique, en vue de l'échange de données. C'est également un formalisme de *spécification*, c'est à dire qu'il permet une description complètement non ambiguë et traitable par machine.

Formalisme de modélisation : langage permettant de représenter formellement des informations et de décrire plus ou moins formellement leur représentation sous forme de données.

Inférence : opération logique portant sur des propositions tenues pour vraies (les prémisses) et concluant à la vérité d'une nouvelle proposition en vertu de sa liaison avec les premières.

Information : connaissance sur un fait, un concept ou un processus.

Information : résultat de la mise en relation de plusieurs données. C'est encore un ensemble de données qui ont été consignées, classées, organisées, raccordées ou interprétées dans un cadre qui en dégage le sens.

Modèle [Habrias97] **:** abstraction de la réalité réalisée en utilisant des concepts mathématiques (ensembles, relations, graphes, …).

Ontologie : spécification formelle consensuelle et référençable d'une conceptualisation d'un domaine.

Propriétés : voir attributs.

Raisonnement : processus qui permet d'obtenir de nouveaux résultats ou de vérifier un fait en faisant appel à différentes lois .

Taxonomie : hiérarchie de spécialisation.