République Algérienne Démocratique et Populaire
Ministère de l'Enseignement Supérieur et de la Recherche Scientifique
Université Mentouri – Constantine –
Faculté des sciences de l'ingénieur
Département Informatique

N° d'ordre :

Série :

Mémoire

Présenté en vue de l'obtention du diplôme de

Magister en Informatique

Option : Systèmes distribués

Modélisation et stratégies de recherche sémantique dans les bases de données XML natives

Présenté par : Mr. Tarek BOURBIA.

Dirigé par : Pr. Mahmoud BOUFAÏDA.

Soutenue-le : 17/05/2009

Devant le jury composé de:

Président :

Zizette BOUFAÏDA Professeur Université Mentouri - Constantine.

Rapporteur :

Mahmoud BOUFAÏDA Professeur Université Mentouri - Constantine.

Examinateurs :

Ramdane MAAMRI Maître de conférence Université Mentouri de Constantine

Fouzia BENCHIKHA Maître de conférence Université de Skikda

REMERCIEMENTS

Je présente mes sincères remerciements pour mon directeur de thèse, Monsieur le Professeur Mahmoud Boufaida pour son aide et ses conseils très avisés tout au long de ma formation.

Je lui témoigne ma profonde gratitude et ma reconnaissance. Son expérience et ses compétences ont facilité mon travail et m'ont profondément marqué.

Je tiens à remercier très sincèrement l'ensemble des membres du jury qui me font le grand honneur d'avoir accepté de juger mon travail.

Je remercie Madame Boufaida Zizette Professeur à l'université de Constantine pour l'honneur qu'elle me fait en acceptant la présidence de ce jury. Qu'il trouve donc ici l'assurance de ma profonde gratitude.

Je tiens à exprimer toute ma reconnaissance à Ramdane Maamri Maitre de conférences à l'université de Constantine, ainsi que à Madame Fouzia Benchicka Maître de Conférence à l'université de Constantine, pour avoir accepté d'être examinateur de ce travail et pour le temps qu'ils ont investi à l'évaluation malgré leurs nombreuses obligations.

Aussi, je remercie tout particulièrement mes parents et ma femme pour leur encouragement et leur soutien surtout dans les moments difficiles. Qu'ils trouvent ici toute ma gratitude.

A mon fils

A mes sœurs et mes frères

A tous mes amis (es) et mes collègues

Avec toute mon affection.

Sommaire

Introduction générale

Chapitre I : Notions sur XML et bases de données XML natives

Chapitre II : Ontologie et bases de données à base ontologique

Chapitre III : Un modèle architectural sémantique pour les bases de données XML natives

Chapitre IV : Implémentation d'une simulation de transformation du modèle de données sémantique OWL-DL vers XML Schéma

Table de figures

Introduction générale

1. Contexte

Les données et les bases de données jouent un rôle primordial dans le développement des technologies de l'information notamment dans le domaine du Web. Comprendre et faire évoluer les fonctionnalités et l'expressivité offertes par les bases de données deviennent une nécessité.

La modélisation des bases de données est en constante d'évolution depuis la première et la deuxième génération (vers 1970) des systèmes de gestion de base de données réseaux et hiérarchiques, dans lesquels les programmes sont dépendants de l'organisation physique des fichiers sur le disque et les applications doivent être modifiées à chaque réorganisation physique. La troisième génération de ces systèmes (vers 1980) avec le modèle relationnel qui représente toutes les informations sous forme de tables. La simplicité de ce modèle a permis la définition de langages de requêtes simples, faciles d'utilisation et puissants (SQL). Les années 1990 ont vu apparaître de nouvelles applications (CAO, PAO, multimédia, Web,...) nécessitant l'utilisation de bases de données. Ces nouvelles applications ont révélé les insuffisances du modèle relationnel. La quatrième génération orientée-objet des modèles de base de données est la solution aux problèmes suscités lors du développement de ces nouvelles applications. Les avantages de la programmation par objet sont reconnus : puissance des concepts, programmation modulaire, réutilisation du code et maintenance facile du code. Dans le but de profiter d'une part, du succès commercial et de la simplicité de format physique de données du modèle relationnel, et d'autre part de l'expressivité et l'extensibilité du modèle conceptuel en objet, un autre modèle est apparu qui est le modèle relationnel-objet.

Avec l'avènement des applications orientées WEB, d'autre outils de représentation des données sont apparus notamment le langage à base de balise XML, et ses modèles de persistance de données, et l'ensemble de bases de données XML et de bases de données XML natives.

XML a été développé pour traiter syntaxiquement de documents. Il est considéré comme une référence de base et la première pierre pour les langages du Web sémantiques. Il permet

également de définir d'autres langages, afin de donner du sens aux données. Parmi ces langages dérivés, le langage d'ontologie Web OWL qui est conçu pour décrire et représenter un domaine de connaissances spécifiques, en définissant des classes de ressources ou objets et leurs relations. OWL permet également d'affirmer des propriétés des objets et de raisonner sur des classes et des individus dans la mesure où il supporte une sémantique formelle du langage OWL.

Les travaux menés en modélisation des connaissances faisaient émerger la notion de l'ontologie OWL. Les derniers travaux menés dans le domaine des bases de données visent à augmenter la capacité de représentation de l'information avec des modèles plus expressifs et riches. Le contexte de notre travail se situe dans l'intersection de deux domaines : les bases de données et les ontologies OWL.

2. Problématique et objectif

Les bases de données XML natives représentent un référentiel intéressant pour les documents et données structurés ou semi structurés. Néanmoins les performances des systèmes de gestion des bases de données XML natifs sont actuellement en cours d'amélioration.

Plusieurs langages de représentation de l'information sémantique, plus sophistiqués que XML, ont été développés afin d'améliorer la recherche sémantique et de raisonner sur les contenus des documents et de données. Le standard OWL (Web Ontology Langage) proposé par le consortium W3C. OWL, est bâti sur la logique de description et s'appuie sur la syntaxe du XML. Le langage d'ontologie OWL fournit une diversité de services pour modéliser, analyser et rechercher des informations, et pour effectuer des inférences. Néanmoins, si la quantité et la taille des données sont importantes, il n'est pas possible de les manipuler en gérant un seul fichier simple, il est nécessaire de les manipuler par un système de gestion de base de données qui permet de représenter l'ontologie en conservant la sémantique et les contraintes. Il faut également stocker les données et les optimiser lors des opérations de recherches et de mises à jours d'une part, et d'autre part les sécuriser et d'assurer leur partage.

Les SGBD XML représentent un niveau syntaxique de définition et de manipulation des données, sans tenir compte de la sémantique. De ce fait, notre travail consiste à étendre les fonctionnalités d'un SGBD XML natif par une couche représentant un niveau sémantique afin d'avoir une représentation plus expressive et de permettre l'inférence sur les bases de données. Cette couche sémantique est décrite à l'aide d'un langage d'ontologie Web OWL.

Pour cela, nous avons défini un ensemble de règles pour transformer une ontologie OWL vers une base de données XML natives en utilisant une description formelle basée sur les schémas XML. Ce mécanisme permet d'exécuter des requêtes de recherches sémantiques basées sur le langage de requête d'ontologie SPARQL pour manipuler l'ontologie OWL dont les instances (données) sont situées au niveau des nœuds feuilles des documents XML.

L'objectif de ce travail consiste, à établir une conception et une modélisation logique sous le langage OWL, et un stockage physique dans une base de données XML native, en vue d'exploiter pleinement la sémantique offerte par le langage d'ontologie OWL, et les fonctionnalités de gestion et manipulation de données offertes par le système de gestion de base de données XML natif.

3. Organisation du mémoire

Ce mémoire, est organisé en quatre chapitres. Les deux premiers présentent l'état de l'art sur les technologies XML, les bases de données XML, l'ontologie OWL et les données à base ontologique. Le troisième est consacré à la présentation de notre contribution, et enfin le quatrième chapitre montre une simulation de l'implémentation du processus de transformation proposé.

- Dans le premier chapitre, nous allons présenter le langage de balises XML en présentant la structure d'un document XML et les technologies avec lequel est attachées (DTD, XML Schema, espace de noms,..). Nous allons expliquer par la suite, dans le même contexte, comment réaliser des requêtes permettant de naviguer et d'extraire des informations particulières dans un document XML par les langages XPATH et XQUERY. A la fin de ce chapitre, nous allons décrire ensuite les deux modes de stockage des documents XML dans les bases de données : la première basée sur l'extension d'un SGBD relationnel ou objet avec une couche XML (base de données XML), et la deuxième consiste à concevoir un SGBD XML spécifique pour l'arbre XML (base de données XML native).

- Le deuxième chapitre sera consacré à l'aspect sémantique de notre travail. Nous décrivons d'abord les langages d'assertions et d'annotations RDF et RDFS qui déterminent les relations entre les ressources ou objets, et qui permettent aussi l'annotation des ressources. Ensuite, nous présentons le langage de présentation des ontologies OWL et sa richesse de présentation des classes et leurs propriétés, et dans le même contexte nous donnons un aperçu sur le langage de requête sémantique SPARQL. Nous terminerons par la définition des bases de données à base ontologique

qui se situent dans l'intersection de deux domaines : les bases de données et les ontologies et une synthèse avec commentaires de quelques travaux dans le domaine des bases de données à base ontologique.

- Dans le troisième chapitre, nous allons présenter notre contribution qui consiste en la proposition d'une architecture conçue suite à l'ajout de la couche sémantique au dessus du SGBD XML natif. L'interaction de cette couche sémantique avec celle syntaxique du SGBD sera illustrée par ses composants : Le schéma de la base de connaissance exprimé à l'aide de la technologie à base ontologique OWL-DL, et le gestionnaire de requêtes sémantiques fonctionnant sur le langage SPARQL. De ce fait, nous allons dénombrer les règles de transformation de l'ontologie OWL-DL vers une base de données XML natives en utilisant une description formelle basée sur les schémas XML. A la fin de ce troisième chapitre, nous présenterons les règles de transformation fondées sur l'utilisation comme intermédiaire l'algèbre relationnelle pour effectuer la correspondance entre les clauses de deux langages de requêtes SPARQL et XQUERY.

- Le quatrième chapitre, est dédié à la présentation de l'implémentation de la simulation de transformation du modèle de donnée sémantique OWL-DL vers XML Schema du niveau syntaxique de la base de données XML native. Nous utilisons le langage de transformation XSLT pour concrétiser la correspondance des différents concepts de l'ontologie vers les différents éléments appropriés du schéma XML cible de la base de données.

En conclusion, nous discutons quelques perspectives dégagées de notre travail pour faire évoluer la technologie sémantique dans le domaine de base de données.

Notions sur XML et bases de données XML natives

1. Introduction

Le Consortium W3C a conçu le langage XML pour répondre aux problématiques d'échanges et d'interopérabilité sur le Web. XML permet de séparer le contenu d'un document de sa forme afin de faciliter la réutilisation du contenu pour différentes fins, en appliquant différents styles de présentation, et en filtrant l'information. XML est un sous ensemble de SGML, il simplifie SGML en reprenant la majorité de ses fonctionnalités.

XML est un métalangage permettant de définir d'autres langages, c'est-à-dire définir de nouvelles balises permettant de décrire d'autres types de documents, XML est la première pierre pour les langages sémantiques, il s'impose ces dernières années comme langage d'intégration et d'échange données et méta données.

Et vu la nécessité de stockage et l'interrogation des documents XML, des SGBD spécifiques sont conçus et implémentés pour assurer la persistance des documents et données XML afin de profiter de l'indexation, d'un moteur de requête, le partage, la sécurité, etc.

Nous présentons dans ce chapitre le langage XML en présentant la structure d'un document XML et les technologies avec lequel est attachées. Nous allons aussi présenter les deux approches de stockage des documents XML dans les bases de données ; la première c'est par l'extension d'un SGBD relationnel ou objet avec une couche XML, et la deuxième consiste à concevoir un SGBD XML spécifique pour l'arbre XML.

2. Présentation du langage XML

XML [2] est conçu pour donner la possibilité de traiter syntaxiquement et d'une manière automatique les documents. La puissance de XML réside dans sa capacité de pouvoir décrire

n'importe quel domaine de données grâce à son extensibilité. Il va permettre de structurer, donner le vocabulaire et la syntaxe des données qu'il va contenir sans se soucier de la présentation, ce qui permet d'afficher un même document sur des applications différentes.

2.1. Structure d'un document XML

XML est un langage de balises comme HTML mais il est extensible, évolutif par ses éléments et attributs qu'ils constituent le document XML, comme il est illustré dans la figure (fig1.1). En XML, les balises ne sont pas prédéfinies. Alors que le HTML a été conçu pour afficher de l'information, le XML a été créé pour structurer de l'information. Un document XML est conforme avec un document SGML (Standard Generalized Markup Language) [2], mais avec plus de simplicité.

Pour qu'un document XML soit bien formé [3], il faut qu'il soit conforme aux règles en ce qui concerne les éléments et les attributs.

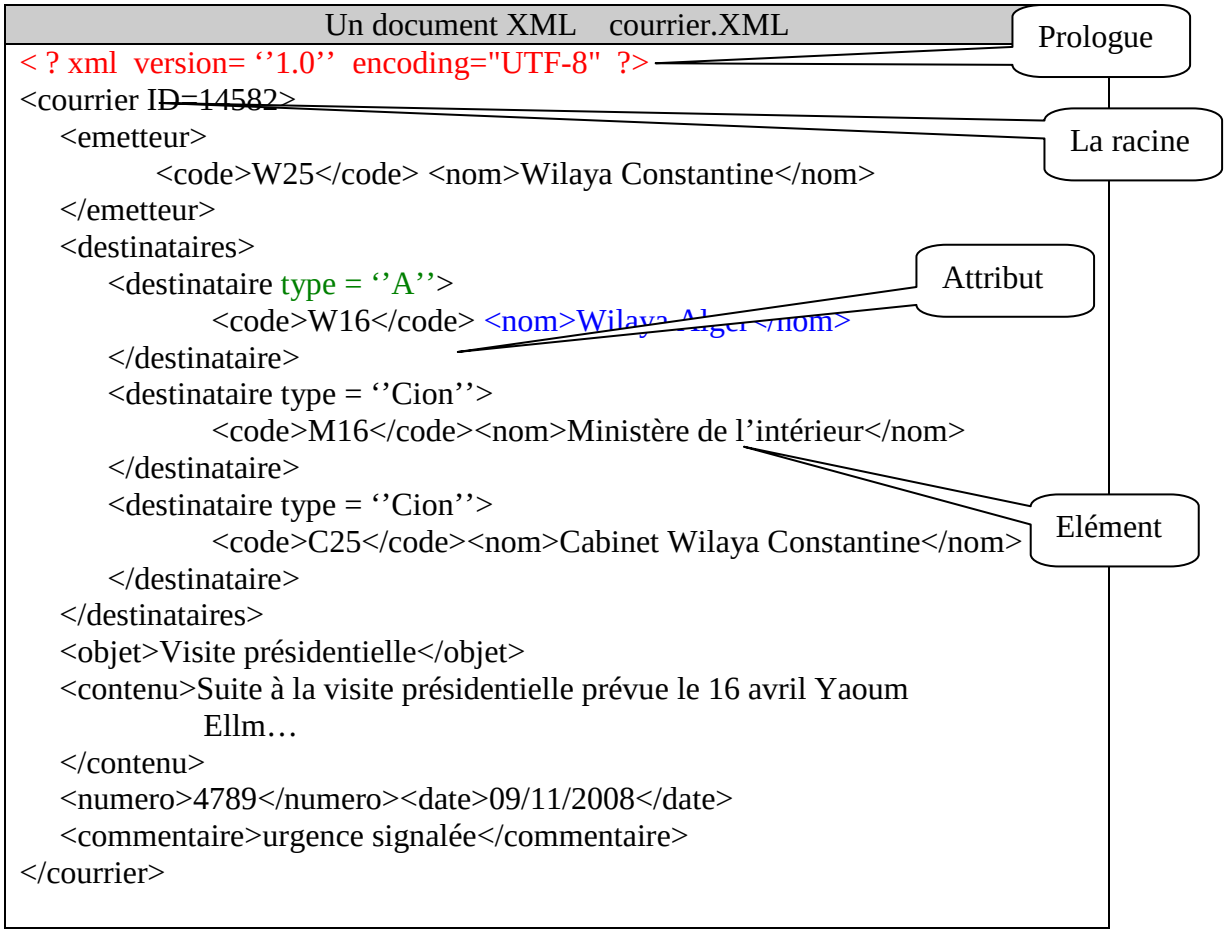


Fig 1.1. Structure d'un document XML

Après le prologue, se trouve une série d'éléments. Un élément est l'unité de base d'un document XML, composé de données textuelles et de balises. Les frontières d'un élément sont définies par une balise de début et une balise de fin. Les balises en XML, au contraire de l'HTML, n'ont pas de signification précise, elles sont présentes pour délimiter les données et XML laisse l'entière interprétation des données à l'application qui les lit. Un document XML bien formé doit contenir exactement un élément non vide, appelé élément racine. Celui-ci peut contenir d'autres éléments définissant ainsi un arbre.

Les éléments peuvent contenir des attributs qui fournissent des informations supplémentaires. Ces attributs sont des couples de la forme nom="valeur". Ils doivent se trouver dans la balise ouvrante après le nom de l'élément. Les valeurs des attributs se trouvent obligatoirement entre guillemets. Comme le précise la spécification XML, tout texte qui n'est pas du balisage est une donnée textuelle du document.

2.2. Technologies attachées à XML

Avec XML, nous pouvons développer des applications qui traitent le contenu des documents XML de manière dynamique en l'adaptant plus facilement à nos besoins. Mais pour le faire il faut s'en servir de plusieurs technologies attachée comme : DTD, XML Schema, XSLT,XPATH, XQUERY, etc.

2.2.1. DTD (Document Type Definition)

Sur le plan conceptuel, un document XML a deux parties : le document XML qui est obligatoirement présent, et une description formelle optionnelle, appelée Document Type Definition (DTD). Cette description contient la liste des éléments organisés, qui constitue la structure d'un document, et les règles syntaxiques que doit respecter le document.

Un document XML accompagné de son DTD, doit se conformer strictement aux règles de cette grammaire. On dit qu'il est valide lorsque la syntaxe et la structure du document XML sont vérifiées par la grammaire imposée par son DTD.

L'avantage [5] est que la DTD permet aux groupes indépendants des utilisateurs de se convenir à se référencer à une DTD commune pour échanger les données.

Une DTD [12][2] définit la structure du document à l'aide d'une liste d'éléments légaux. Le mot-clé !ELEMENT définit le type de l'élément et le mot-clé !ATTLIST précise ses attributs. La DTD permet également de déclarer le nombre de fois qu'un élément fils peut apparaître dans un élément parent : un ou plus (+), zéro ou plus (*) ou zéro ou un (?).

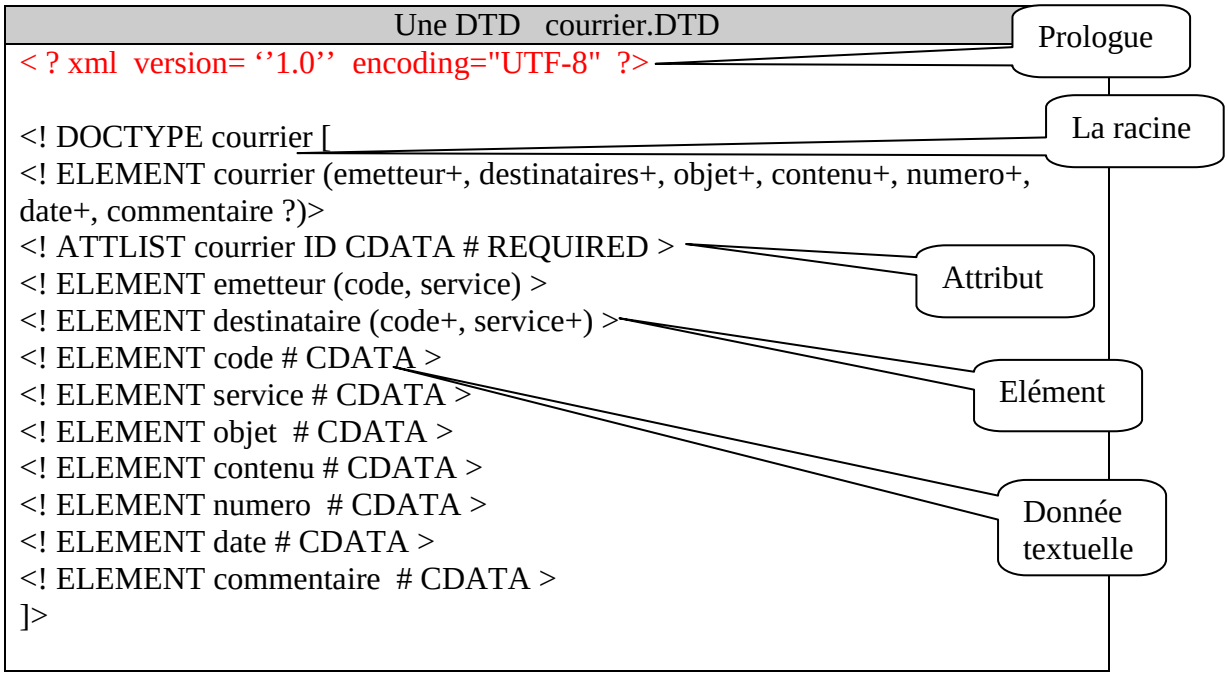


Fig 1.2. DTD

Les attributs de type ID et IDREF permettent de créer des relations entre les attributs. Les types d'éléments possibles sont les suivants :

- #PCDATA désigne les données textuelles devant être traitées par l'analyseur ;
- #CDATA désigne les données textuelles qui ne doivent pas être traitées par l'analyseur ;
- EMPTY signifie que l'élément ne peut rien contenir ;
- ANY signifie que l'élément peut contenir ce que l'on veut.

Pour illustrer, voire l'exemple de la figure (fig 1.2)

Cependant, un des gros défauts des DTD, malgré leur simplicité, est qu'il n'est pas possible de définir des contraintes comme le nombre de fois qu'un élément particulier doit apparaître dans le document, le type des données de chaque élément, les DTD ne sont pas au format XML et ne supporte pas les espaces de nom.

Des contraintes d'instanciation et de typage sont moins critiques pour des documents XML orientés présentation.

2.2.2. XML Schema

Le schéma XML [12] est un peu équivalent à celui d'une DTD; c'est-à-dire qu'il permet de valider un document XML. XML Schéma comporte cependant quelques différences : les types de données de base utilisables dans les DTD (#PCDATA, ANY, EMPTY) ont été enrichis (entiers, réels, chaîne, date, liste, . . .). Les types de données sont dérivables : on peut donc réutiliser un type défini pour l'enrichir. Il est également possible de grouper les attributs en factorisant leurs définitions afin de faciliter leur réutilisation. L'on peut également définir précisément le nombre d'occurrences d'un élément au sein d'un document XML.

Il est intéressant de remarquer que XML Schéma est lui même défini par un schéma, dont les balises de définition s'auto-définissent, ce qui en fait un exemple de définition récursive.

Le langage de définition des schémas XML décrit les éléments et les attributs destinés à être utilisés dans des documents XML, comme il est illustré dans la figure (fig1.3.), les différences entre les types simples et complexes, la manière de définir des types complexes, l'utilisation des types simples comme valeurs d'éléments et d'attributs, l'annotation des schémas, ainsi que les mécanismes de dérivation des types à partir des types fournis par XML Schema et de contrôle de ces dérivations.

Le terme d'instance de document est souvent utilisé pour désigner un fichier contenant un document XML conforme à un schéma particulier.

Nous disons que les éléments qui contiennent des sous-éléments ou portent des attributs sont de type complexe, tandis que les éléments qui contiennent des nombres (ou des chaînes de caractères, des dates, etc.) sans autre sous-élément sont de type simple. Quelques éléments ont des attributs; les attributs sont toujours de type simple.

Le schéma de courrier courrier.XSD

```
<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema">
 <xsd:annotation>
  <xsd:documentation xml:lang="fr">
   schéma du document courrier créé par BOURBIA Tarek 09/11/07…
  </xsd:documentation>
 </xsd:annotation>

 <xsd:element name="courrier" type="CourrierType"/>

 <xsd:element name=" commentaire " type="xsd:string"/>

 <xsd:complexType name="CourrierType">
  <xsd:sequence>
   <xsd:element name="emetteur" type="service"/>
   <xsd:element name="destinataires" type="Destinatires"/>
   <xsd:element name="objet" type="xsd:string"/>
   <xsd:element name="contenu" type="xsd:string"/>
   <xsd:element name="numero" type="xsd:myInteger"/>
   <xsd:element name="date" type="xsd:date"/>

   <xsd:element ref="commentaire" minOccurs="0"/>
    </xsd:complexType>

 <xsd:complexType name="service">
  <xsd:sequence>
   <xsd:element name="code"   type="xsd:string"/>
   <xsd:element name="nom" type="xsd:string"/>
  </xsd:sequence>
</xsd:complexType>

 <xsd:complexType name="Destinataires">
  <xsd:sequence>
   <xsd:element name="destinataire" type =  minOccurs="1"
      maxOccurs="unbounded">
     <xsd:attribute name="type" type=="service"  use="required"/>
  </xsd:complexType>

</xsd:schema>
```

Fig 1.3. Schéma xml

Les composants de bases du langage de définition des schémas XML sont :

a) Type simple

Le schéma de notre courrier fait appel à plusieurs éléments et attributs de types simples. Quelques uns de ces types simples, tels que **string** et **date**, sont prédéfinis dans XML Schema

tandis que d'autres sont dérivés de ces types pré-éxistants. Par exemple, l'élément numero a un type appelé **myInteger** (fig1.3.), qui est dérivé du type simple de base **integer**.

b) Contrainte d'occurrences

Dans, le complexe type Destinataires, l'élément destinatiare est défini comme étant obligatoire par le biais de la valeur 1 de l'attribut **minOccurs**. En général, un élément est optionnel quand la valeur de **minOccurs** vaut 0 ou plus. Le nombre maximum de fois qu'un élément peut apparaître est déterminé dans sa déclaration par la valeur de l'attribut **maxOccurs**, ou encore le mot unbounded qui signifie qu'il n'y a pas de valeur limite. La valeur par défaut dans les deux cas (**minOccurs** et **minOccurs**) est 1.

c) Type complexe

Les types complexes sont créés en utilisant l'élément **complexType** composé d'une série de déclarations d'éléments et d'attributs et de références d'éléments. Les éléments sont déclarés en utilisant l'élément **element** et les attributs sont déclarés en utilisant l'élément **attribute**. Par exemple, l'élément Service est défini comme étant de type complexe, comme il est illustré dans la figure (fig1.3.), et à l'intérieur de sa définition, nous remarquons la présence de deux déclarations d'éléments.

2.2.3. XLINK et XPointer

Les documents XML sont liés entre eux par des liens. XLink [23] décrit une méthode pour ajouter des liens hyper-textes à un fichier XML. XPointer [24] permet de se référer à des parties de document XML.

Un lien XML est la spécification (insérée dans le document) d'une association explicite entre des ressources ou portions de ressources, ainsi que de l'information descriptive définie par XLink. XLink permet des liens entre plus de deux ressources, permet d'associer des métadonnées avec un lien, de relier des bases de données qui sont en un emplacement différent des ressources auxquelles elles sont liées, et de créer des hypertextes.

HTML a le « *A element* » qui permet de créer des liens avec les URI, il faut aller plus loin, permettre une modélisation de liens. XLink sert à identifier des liens simples et étendus, à décrire les rôles des ressources dans la relation, ainsi qu'à regrouper des extrémités.

Le langage XPointer est utilisé pour identifier les fragments pour toute référence URI qui localise une ressource de l'Internet dont le « type media » est *text/xml* et *application/xml*. XLink s'occupe de la localisation pour les autres types de ressource. XPointer est basé sur XPath et le XML Information Set, et il sert à l'adressage dans les structures internes de

documents XML. Il permet des traversées et des choix en fonction de diverses propriétés comme les types d'élément, les valeurs d'attribut, le repérage dans les données caractère, la position relative. La sélection de portions d'arborescence se fait grâce aux axes, aux prédicats et aux fonctions.

L'identifiant de fragment Xpointer peut s'écrire au long (*full Xpointer*), le nom seul (*bare name*), la séquence des enfants (*child sequence*). Le jeu de caractères est l'Unicode comme XML. Alors que XPath se limite ses localisations aux noeuds et ensemble de noeuds, Xpointer inclut les points et les étendues comme localisations possibles, avec le même sens que dans DOM2.

2.2.4. Espaces de nommage

Un document XML peut utiliser plusieurs DTD ou schémas XML. Et par conséquent des conflits sur les noms utilisés peuvent apparaître. Si un même élément est défini de manière différente dans deux schémas, l'analyseur doit savoir quel schéma appliquer. Pour cela, les noms sont préfixés par un nom de schéma [12].

L'espace de nommage XML, est une collection des noms, identifiée par une référence URI, qui sera utilisé dans un document XML comme types des éléments et noms des attributs.

La déclaration de l'espace de nommage se fait par la syntaxe suivante :

xmlns:nom="URI" (généralement schéma, c'est l'adresse de l'espace de nommage)

Et son utilisation est comme suit : nom de l'espace de nommage:élément et nom de l'espace de nommage:attribut.

Par exemple : <xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema">

2.2.5. XSL et XSLT

Le navigateur Web ne peut pas interpréter les balises XML car elles sont définies librement par le programmeur. Et pour afficher les documents XML, il faut utiliser les feuilles de style. Soit CSS qui est conçu pour HTML, et soit le langage XSL (eXtensible Stylesheet Language) [20] qui est plus adapté à la technologie XML et donc plus riche et plus complexe que CSS.

En outre, XSL est utilisé autrement pour transformer un document XML vers un autre document à base XML, dans le but d'extraire, de filtrer ou de restructurer les données et les balises. Dans ce sens, XSL est un langage de requêtes pour XML.

Le langage XSL fournit les fonctionnalités suivantes :

- Sélectionner une partie des éléments XML ;
- trier des éléments XML ;
- filtrer des éléments XML en fonction de certains critères ;
- choisir et retenir des éléments par des tests conditionnels.

La richesse des fonctionnalités du XSL est due particulièrement de ces composants. Le premier composant est le langage XSLT (eXtensible Stylesheet Language Transformation) [25] qui transforme un document XML en un autre document XML ou vers d'autres formats. Le deuxième composant est Xpath [21] qui permet de localiser et d'adresser par des chemins des parties ou éléments d'un document XML.

Un programme XSL est un ensemble de règles. Chaque règle est composée de deux parties. Un motif (Pattern), qui décrit la structure des données concernées par cette règle. Et un patron (Template), qui décrit la structure du résultat généré par la règle.

Exemple :

Soit le document XML suivant :

<courriers>

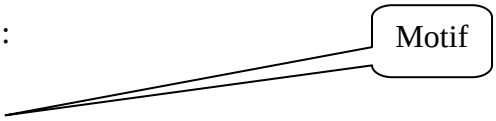
<courrier><objet> RDV</objet><numero-courrier>3455</numero-courrier></courrier>

……

<courrier><objet>Visite Présidentielle</objet><numero-courrier>39</numero-courrier></courrier>

</courriers>

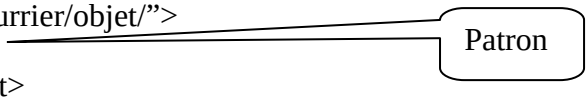
Soit le programme XSL suivant :



<xsl:template>

<xsl:apply-template/> </xsl:template>

<xsl:template match="couriers/courrier/objet/">



<résultat><xsl:value-of/></résultat>

</ xsl:template >

Après application de ce programme XSL sur le document XML précédent, le résultat obtenu est :

<résultat>RDV</résultat>

……

<résultat> Visite Présidentielle</résultat>

Ce programme fournit en sortie l'ensemble des objets de courriers.

2.2.6. Langages de requêtes pour XML

Les langages de requêtes pour XML sont les aspects de traitement, d'interrogation et de manipulation des masses de données lorsque celles-ci sont disponibles au format XML. Ces langages permettent d'accéder à un ensemble de nœuds d'un arbre XML, recherchés selon leur position, leur structure ou leur contenu. Plusieurs langages de requêtes sont développés mais les plus utilisés sont ceux du consortium W3C : XPATH pour trouver des parties d'un document ; XQUERY qui est un langage de requête pour XML, permettant d'interroger de manière complexe ; et XUPDATE du groupe XML:DB pour mettre à jour les valeurs des nœuds dans un document XML.

a) XPATH

XPATH est un langage de navigation qui permet de naviguer dans les éléments, les attributs et les textes d'un document XML et de l'interroger. L'objectif de XPATH [21] est de localiser des parties de documents XML à l'aide d'expression de chemin, En vue de pourvoir à cet objectif premier, XPATH fournit également les moyens pour traiter des chaînes de caractères, des nombres et des booléen. XPATH aussi, est conçu pour être utilisé à la fois par XSLT, XPointer et XQUERY.

Les expressions de XPATH sont similaires de celles utilisées par les systèmes de fichiers, par exemple: /courrier/destinataires/destinataire [type eq "CION"]/nom

XPATH navigue dans le document XML courrier.xml, et sélectionne les noms des services destinataires et qui ont reçu les courriers à titre d'information "Cion".

Les prédicats sont des expressions logiques et doivent être écrits entre crochet et permettent de filtrer les éléments lors du parcours.

Les expressions de chemin sont composées d'étapes. Les étapes sont représentées par un axe, un nom d'élément et éventuellement un prédicat. Les axes permettent de définir le sens de parcours du document XML. Les axes de parcours sont présentés au tableau 1.1.

Axes	Ensemble de nœuds résultant
parent (../)	Le premier nœud sur le chemin de u vers le nœud racine
ancestor	Tous les nœuds sur le chemin de u vers la racine
ancestor-or-self	u et tous les nœuds sur le chemin vers la racine
Child (/)	Les descendants directs du nœud u
descendant	Tous les nœuds dont u est l'ancêtre
descendant-or-self (//)	u et tous les nœuds dont u est l'ancêtre
preceding	Les nœuds précédents le nœud u (excepté les ancêtres) du document
following	Les nœuds suivants le nœud u (excepté les descendants) du document
preceding-sibling	Les frères précédents du nœud u dans l'ordre du document
following-sibling	Les frères suivants du nœud u dans l'ordre du document
attribute	Les attributs du nœud u
Self (./)	Le nœud u

Tableau 1.1. Les axes XPATH

b) XQUERY

XQuery est le langage de requêtes pour XML défini et standardisé par le W3C XQUERY [25]. C'est un langage de requête qui s'appuie sus XPATH, il est conçu pour permettre d'exprimer et réaliser des requêtes précises et compréhensibles. Il introduit un type de requête basé sur des expressions FLWOR pour For, Let, Order, Where et Return. Comparable aux expressions select-from-where de SQL.

FLWOR fournit les cinq clauses principales suivantes:

1. for: mécanisme d'itération sur un ensemble de nœuds. La combinaison de source de données devient triviale et la jointure aussi par for, puisque for autorise l'ouverture de plusieurs documents XML simultanément.;
2. let: permet l'affectation des variables;
3. where: les clauses for et let génèrent un ensemble de nœuds XML qui peuvent être filtrés par des prédicats de la clause where;
4. order by: ordonne les résultat de façon alphabétique ou numérique, ascendant ou descendant.
5. return: construit le résultat pour chaque nœud satisfaisant la clause where. Return et element permettent de mettre en forme un document XML comme résultat et d'exécuter des opérations d'agrégations et de traitements.

Par exemple la requête ci-dessous a pour résultat tous les objets des courriers envoyés à la wilaya d'Alger ordonnés par date d'envoi:

```
for $d in document ("courrier.xml")  /courrier
where $d/destinataires/destinataire/code="W16"
order by $d/date ascending
return $d/objet
```

XQUERY ne couvre pas toutes les fonctionnalités. Notamment, la recherche plein texte et la mise à jour de documents XML.

Aussi, XQUERY, bien qu'il soit un langage d'interrogation pour XML, ne respecte pas la syntaxe XML, XQUERYX possède une syntaxe alternative proposée, pour représenter une requête comme un document XML bien formé. XQuery s'impose comme le langage de requêtes pour les bases de données XML natives.

c) XUPDATE

XUpdate [8] est un langage qui utilise une méthode déclarative pour mettre à jour la valeur des nœuds dans un document XML. XUpdate s'appuie sur les expressions XPath pour sélectionner l'emplacement dans un document XML des noeuds à mettre à jour.

L'approche générale consiste à utiliser un document XML contenant un ensemble de commandes de mises à jour. L'ensemble de commandes de mises à jour sont les suivantes :

insert (insertion), append (ajout en fin de liste), update (mise jour), remove (suppression) et rename (renommer).

Un document XUPDATE est structuré comme suit :

<xupdate:opération select="/location/path">

<...>contenu<...>

</xupdate:opération>

opération : insert-before, insert-after, insert, append, update, remove, rename.

/location/path : le chemin de l'emplacement des nœuds par la notation XPATH.

contenu : les nœuds (élément, attribut, texte)

Exemples :

a- renommage d'un élément et d'un attribut dans le document courrier.XML

<xupdate:rename select="/courrier/date">date_courrier</xupdate:rename>

<xupdate:rename
select="/courrier/destinataires/desinataire/@type">type_destinataire</xupdate:rename>

Après évaluation de ces deux expressions, l'élément date est renommé en date_courrier et l'attribut type est renommé en type_destinataire.

b-modification du contenu d'un élément

<xupdate:update select="/courrier/destinataires[@type_destinataire="Cion"]/@type_destinataire">

Ampliation

</xupdate:update>

Après évaluation de l'expression, le type de destinataire prendra pour valeur : "Ampliation", au lieu de "Cion".

XUpdate est relativement basique et, dans l'état actuel, XUpdate ne propose pas encore la validation de schéma, les traitements conditionnels, la définition de block de transactions.

2.2.7. DOM et SAX

DOM [40] (Document Object Model) définit la structure d'un document XML sous forme d'une hiérarchie d'objets, afin de simplifier l'accès aux éléments constitutifs du document.

Plus exactement DOM est un langage normalisé d'interface (API), indépendant de toute plateforme et de tout langage, permettant à une application de parcourir la structure du document et d'agir dynamiquement sur celui-ci.

SAX est une API basée sur un modèle événementiel, cela signifie que SAX permet de déclencher des événements au cours de l'analyse du document XML. Une application utilisant SAX implémente généralement des gestionnaires d'événements, lui permettant d'effectuer des opérations selon le type d'élément rencontré.

3. Stockage d'un document XML

Le contenu des documents XML, à l'instar des données au format électronique, doit être stocké dans un référenciel comme dans une base de données, afin de mieux le gérer et faciliter son exploitation par les applications.

Il existe deux approches pour le stockage d'un document XML : la première consiste à étendre un SGBD relationnel ou objet avec une couche XML, la deuxième consiste à concevoir un SGBD XML spécifique pour l'arbre XML. Avant de détailler la manière de stockage du contenu d'un document XML dans une base de données, il faut d'abord délimiter les types de ce contenu.

3.1 Types de contenu d'un document XML

Le stockage des contenus des documents XML, dans des bases de données, nécessite de distinguer entre un contenu centré données et un contenu centré document [10][16].

3.1.1 Contenu centré données

Les documents XML centrés données sont conçus pour les stocker et traiter à partir d'une base de donnée. Le stockage relationnel avec middleware de transformation en XML est bien adapté avec ce type de contenu. Par conséquent, le document XML n'est pas conservé (reconstituable) et les données sont extraites des fichiers XML puis stockées dans une BD classique (relationnelle).

Les caractéristiques du contenu centré données sont :

- Une structure régulière ;
- Une granulité fine ;
- Pas de contenu mixte ;
- L'ordre des éléments n'est pas important.

3.1.2 Contenu centré documents

Les documents XML centrés document sont conçus pour les stocker dans leur intégrité et gérer par le SGBD, c'est un stockage natif XML. Les données sont retournées sous forme de documents XML sans middleware. Par conséquent, le document XML extrait de la base identique à l'original et avec rapidité.

Les caractéristiques du contenu centré document sont :

- Une structure irrégulière ;
- Une granulité forte ;
- Contenu mixte ;
- L'ordre des éléments est important.

Il est difficile de distinguer entre un document XML centré données et un document XML centré documents ; un document centré données peut avoir des éléments de granulité forte, comme dans l'exemple de la figure (fig 1.1), l'élément « contenu », qui signifie le corps du courrier, est de granulité forte.

3.2 Bases de données XML

Ce type de bases de données est géré par des SGBD étendus (objet ou relationnel) avec des outils pour le traitement de documents XML[10][16]. Les données sont stockées dans un format autre que le XML. Il faut donc mettre en place un processus de conversion de schéma (modèle de description de données) du format actuel de la base en documents XML, puis du XML au format utilisé pour le traitement des données, cela influence les performances de traitement des données.

Donc, un mécanisme de transformation assure le passage du schéma XML (DTD, XML Schema) vers le schéma de la base de données (ensemble de tables relationnelles) et vice versa. L'extraction d'un document XML décomposé devra être recomposé par ce mécanisme de transformation qui s'appelle XML publishing[10][16].

La structure XML est un arbre ordonnée, avec une structure irrégulière (éléments et attributs optionnels, éléments multiples). Par revanche, le modèle relationnel est un modèle ensembliste (table et n-uplet d'enregistrements)

Des outils sont disponibles avec les bases de données relationnelles pour traiter ce genre de tâche, comme Oracle9i et IBM DB2. Ils peuvent traduire en XML les données, structurées ou

non. Toutefois, si ces outils sont utilisés pour traduire des données en direct lors d'un échange XML, cela risque d'augmenter les temps de traitement des transactions XML et de ralentir également les autres applications qui accèdent à la base de données relationnelles.

3.2.1 Transformation basée sur l'objet/relationnel

Avec la transformation basée sur des objets, le processus transforme et faire correspondre le document XML à la base de données selon un schéma relationnel-objet dans lequel les types d'éléments sont généralement considérés comme des classes, les attributs et les éléments PCDATA étant traités comme des propriétés de ces classes. Cette approche modélise le document XML sous forme d'arbre d'objets spécifiques des données du document; ces objets sont ensuite mappés à la base de données relationnelle tel que les classes correspondent à des tables et les propriétés scalaires à des colonnes [10][16].

3.2.2 Transformation basée sur les tables

Avec la transformation basée sur les tables relationnelles, le processus transforme les données entre un document XML et une base de données relationnelle. Il modélise les documents XML sous une forme d'une table ou un ensemble de tables. Cela signifie que la structure d'un document XML doit être comme suit :

```
<base de données>
    <table1>
        <ligne1>
            <colonne1>…</colonne1>
              <colonne2>…</colonne2>
            ……
        </ligne1>
        ……..
    </table1>
……
</base de données>
```

L'inconvénient de cette transformation est qu'elle ne peut pas être utilisée pour un document qui n'est pas conforme au schéma exposé ci-dessus.

3.2.3 Génération de schéma relationnel à partir d'un schéma XML

La manière de générer de schéma relationnelle à partir de schéma XML et vice-versa consiste à coder un chemin vers la correspondance basée sur un modèle objet relationnel, qui possède un certain nombre de caractéristiques optionnelles. Des procédures similaires existent pour les bases orientées objet.

Pour générer un schéma relationnel à partir d'un schéma XML, il convient de :

1. Créer une table et une colonne clé primaire pour tout type d'éléments complexes.
2. Pour chaque type d'élément possédant un contenu mixte, créer une table séparée dans laquelle sont stockées les PCDATA ; cette table est liée à la table parente grâce à la clé primaire de celle-ci.
3. Pour chaque attribut de ce type d'élément qui possède une valeur unique, et pour chaque élément fils simple présentant une seule occurrence, créer une colonne dans cette table. Si le schéma XML contient des informations concernant le type de données, affecter le type de données de la colonne au type qui lui correspond. Dans le cas contraire, affecter lui un type prédéterminé comme CLOB ou VARCHAR(255). Si le type de l'élément fils ou de l'attribut est optionnel, attribuer à la colonne la possibilité d'y affecter des valeurs nulles.
4. Pour chaque attribut possédant plusieurs valeurs et pour chaque élément-fils simple présentant plusieurs occurrences, créer une table séparée pour stocker des valeurs ; cette table est liée à la table parente grâce à la clé primaire de celle-ci.
5. Pour chaque élément fils complexe, lier la table du type d'élément parant à la table du type de l'élément fils à l'aide de la clé primaire de la table parent.

Pour générer un schéma XML à partir d'un schéma relationnel, il convient de :

1. Créer un type d'élément par table.
2. Pour chaque colonne de cette table qui ne soit pas une clé et pour les colonnes correspondants à la clé primaire, ajouter un attribut au type d'élément ou ajouter un élément fils de type PCDATA seul à son modèle de contenu.
3. Pour chaque table pour laquelle la clé primaire est exportée, ajouter un élément fils au modèle de contenu, puis traiter la table récursivement.

4. Pour chaque clé étrangère, ajouter un élément fils au contenu du modèle et traiter récursivement la table de la clé étrangère.

Ces procédures présentent un certain nombre d'inconvénients. Plusieurs sont faciles à corriger manuellement, comme par exemple les conflits de noms et la spécification des types de données et des longueurs. (Les DTD ne contiennent pas d'information sur les types de données ; il est donc impossible de prévoir quels types de données devraient être utilisés dans la base. On notera que les types de données et les longueurs peuvent être déduits d'un document XML Schema.)

Un problème plus sérieux existe : le "modèle" de données utilisé par le document XML est souvent différent (et habituellement plus complexe) que le plus efficace des modèles de stockage de données dans une base. Considérons par exemple le fragment suivant de document XML :

```
 <Client>
<Nom>ABC Industries</Nom>
<Adresse> <Rue>123 Main St.</Rue>
  <Ville>Fooville</Ville>
  <Etat>CA</Etat>
  <Pays>USA</Pays>
  <CodePostal>95041</CodePostal>
</Adresse> </Client>
```

La procédure permettant de générer un schéma relationnel à partir d'un schéma XML conduirait à créer ici deux tables : une pour les clients et une pour les adresses. Dans la plupart des cas, pourtant, il serait plus intelligent de stocker l'adresse dans la table client et non dans une table séparée.

3.2.4 Principaux SGBD XML

Il existe des SGBD XML, s'appuyant sur les techniques de transformation entre le schéma XML et les tables relationnelles.

Le tableau suivant présente les principaux SGBD XML :

Éditeur	Produit	Caractéristiques	Référence
IBM	DB2 UDB - XML Extender	Base de données relationnelle offrant trois possibilités de stockage en XML et des outils de mapping pour effectuer la conversion entre les deux univers. Interrogation en SQL 3 et Xpath. Intégration des schémas XML prévue.	www-01.ibm.com/software/data/db2/support/xmlextender
Microsoft	SQL Server SQLXML 4.0	Base de données relationnelle ne prenant pas en compte le stockage natif XML. Microsoft complète sa solution avec SQLXML, un kit de développement pour des applications fondées sur le langage XML. L'ensemble autorise le résultat de requêtes SQL en XML, la production de documents bien formés et valides, la mise à jour de données relationnelles à partir de contenus XML.	www.microsoft.com
Oracle	Oracle 9i	Base de données relationnelle offrant trois possibilités de stockage en XML et des outils de mapping pour effectuer la conversion entre les deux univers. Interrogation en SQL 99 et Xpath. Intégration des schémas XML prévue.	www.oracle.com
Sybase	Adaptive Server Enterprise	Base de données relationnelle offrant les possibilités de gérer les documents XML de le stocker et d'assurer aussi l'intégration.	www.sybase.com

Tableau 1.2. Principaux SGBD XML

3.3 Bases de données XML natives

L'une des définitions possibles de ce type de base de données, donnée par les membres de l'organisation XML:DB[1], est la suivante :

> Une base de données XML native définit un modèle logique de document XML, stocke et retrouve les documents en fonction de ce modèle. Le modèle doit au minimum inclure les éléments, les attributs, les PCDATA et l'ordre interne du document.
>
> Le document XML est l'unité fondamentale du stockage (logique) dans une base de données XML native, tout comme une ligne d'une table constitue l'unité fondamentale du stockage (logique) dans une base relationnelle.
>
> Une base de données XML native ne repose pas sur un modèle physique particulier pour le stockage. Elle peut par exemple être bâtie aussi bien sur une base relationnelle, hiérarchique, orientée-objet, ou bien utiliser des techniques de stockage propriétaires comme des fichiers indexés ou compressés.

Par conséquent, Les bases de données XML natives [10] [16] sont conçues spécifiquement pour XML. Elles permettent de stocker les documents XML dans leur intégralité, sans les décomposer en éléments, sous une forme de forêts d'arbres, comme il est illustré dans la figure (fig 1.4) [26]. Le document XML est l'entité centrale de la base (comme une relation dans une BD relationnelle) [26]. La majorité des SGBD XML natif utilise le modèle de données XML Schéma [12] [28] comme modèle logique de définition de donnée.

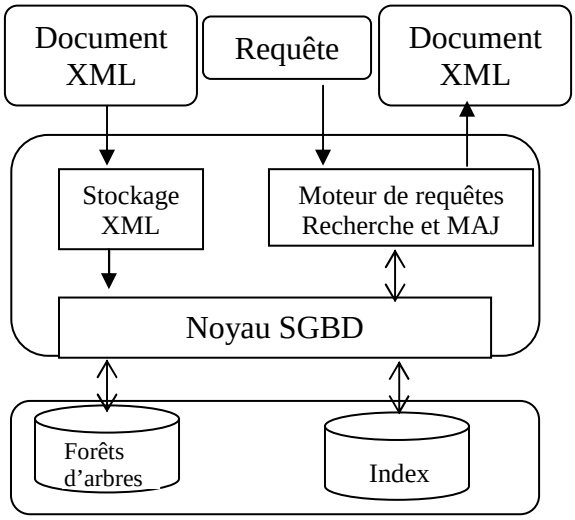


Fig 1.4. Architecture commune des SGBD XML natifs

[1] http://xmldb-org.sourceforge.net/

L'existence d'une base de données XML native n'est pas indispensable pour stocker des documents XML, néanmoins ce type base de données est intéressant lorsque nous cherchons des performances meilleures. Les SGBD XML natifs permettent de manipuler la structure arborescente d'un document XML sans transiter par les transformations.

3.3.1 Stockage des données dans les bases de données XML natives

Il existe deux grandes catégories d'architectures pour le stockage des données dans les bases de données XML natives : les architectures basées sur le texte et celles qui sont basées sur un modèle.

a) Stockage basé sur le texte

Une base de données XML native basée sur le texte stocke le XML en tant que texte. Cela peut être un fichier dans un système de fichiers, un BLOB dans une base relationnelle, ou un format propriétaire.

Les index sont communs à toutes les bases de données XML natives basées sur le texte. Ils permettent au moteur de recherche de naviguer facilement en tout point d'un document XML quelconque. Cela procure à ce genre de base un avantage considérable en matière de vitesse quand on recherche des documents entiers ou des fragments de documents ; la base peut en effet réaliser une seule consultation de l'index, positionner la tête de lecture du disque une seule fois, puis, en supposant que le fragment requis est stocké dans des octets contigus sur le disque, retrouver le document entier ou un fragment en une seule lecture. Au contraire, réassembler un document à partir de morceaux comme on le fait avec une base relationnelle.

b) Stockage basé sur un modèle

Plutôt que de stocker un document XML en tant que texte, Ce type de stockage construit un modèle objet interne du document et stockent ce modèle. La manière dont le modèle est stocké dépend de la base. Certains produits stockent le modèle dans une base relationnelle ou orientée objet. Stocker par exemple le DOM [40] dans une base relationnelle pourrait conduire à des tables du genre Éléments, Attributs, PCDATA, Entités et Références des Entités. D'autres bases utilisent un format de stockage propriétaire adapté à leur modèle.

Les bases XML natives basées sur un modèle et construites sur d'autres bases possèdent vraisemblablement des performances similaires à ces bases sous-jacentes lors de la recherche des documents, et ce, pour la raison évidente qu'elles reposent sur ces systèmes pour retrouver les données. Cependant, la conception de la base laisse place à des variations. Par exemple, à partir d'une base utilisant une stricte correspondance du DOM avec un modèle

objet relationnel, il pourrait en résulter un système qui sollicite l'exécution d'instructions SELECT distinctes pour retrouver les enfants de chaque nœud. D'un autre côté, la plupart des bases de ce genre optimisent leurs modèles de stockage et leur logiciel de recherche.

Lorsque l'on recherche les données dans l'ordre où elles sont stockées, les bases XML natives basées sur un modèle et qui utilisent un format de stockage propriétaire possèdent vraisemblablement des performances similaires aux bases XML natives basées sur le texte. Ceci est dû au fait que la plupart de ces bases utilisent des pointeurs physiques entre les nœuds, ce qui devrait fournir des performances similaires à la recherche textuelle. Les systèmes basés sur le texte sont manifestement plus rapides pour renvoyer des documents textuels, tandis que les systèmes basés sur un modèle sont indubitablement plus rapides pour renvoyer des arbres DOM.

3.3.2 Caractéristiques des bases de données XML natives

Pour gérer efficacement le contenu des documents XML, il est impératif que certaines caractéristiques et fonctions soient disponibles dans une base de données XML native [16] comme :

a) Groupes de documents

Un groupe de document dans une base de données XML native facilite et restreint la recherche sur une collection parmi les collections existantes. Comme une table dans le modèle relationnelle.

b) Langages de requête d'extraction et de mises à jour

Xpath et XQUERY du W3C sont les langages de requêtes de recherche supportés par presque toutes les bases de données XML natives. Quant à les mises à jour c'est XUPDATE du groupe XML :DB qui est le plus supporté; Toutefois, plusieurs autres langages de requête existent.

c) Transactions et accès concurrentiels

Les SGBD XML natives doivent assurer les transactions avec le principe de validation et d'annulation. Et pour assurer une gestion d'accès concurrents, un mécanisme de verrouillage doit être appliqué au niveau des fragments du document XML petits dans le but d'accélérer les accès multiutilisateurs.

d) Middleware

Les SGBD XML natifs doivent fournir pour les applications, en addition aux fonctionnalités internes, une interface middleware avec des méthodes permettant la connexion à la base de données, la manipulation des éléments XML, l'exécution des requêtes de la recherche et de mise à jour avec retour les résultats.

e) Aller-retour des documents

Ce terme signifie la possibilité de pouvoir enregistrer un document et de pouvoir le restaurer par la suite dans un état strictement identique. Pour certaines applications pratiques, il est absolument nécessaire de pouvoir retrouver un document tel qu'on l'avait construit au départ. De plus, comme les documents XML contiennent leur propre définition, le Schéma XML ou la DTD qui leur est associé peut ne plus les reconnaître et en déduire qu'ils ont été corrompus. Cela signifie que les documents ne pourront plus être correctement interprétés par la suite.

f) Importation des données

Quelques SGBD XML natifs peuvent inclure un mécanisme d'importation des données distantes et de les faire stocker dans les documents XML de la base. Ce sont habituellement des données retrouvées à partir de bases relationnelles, et qui sont élaborées en utilisant la correspondance basée sur une table ou la correspondance basée sur un modèle objet relationnel.

g) Indexation

Les SGBD XML natifs supportent l'indexation des valeurs des éléments et des attributs. Les index sont utilisés pour accélérer les recherches, comme pour les bases de données non XML.

h) Intégrité référentielle

L'intégrité référentielle est bien assurée dans une base de données relationnelle par les clés étrangères. Dans les SGBD XML natifs, l'intégrité référentielle consiste à vérifier que tous les liens pointent sur des documents ou des fragments de documents valides. Ces fonctions de contrôle ne sont disponibles que sur certaines bases de données natives XML.

3.3.3 Principaux SGBD XML natives

Il existe déjà des SGBD XML natifs, conçus spécialement pour gérer les documents XML, tout en offrant des fonctionnalités qui facilitent l'exploitation des données et éléments des

forêts d'arbres constituant la base de données. Le tableau suivant présente les principaux SGBD XML natifs :

Éditeur	Produit	Caractéristiques	
dbXML Group	dbXML Core 4	Stockage natif XML, commercialisée sous Licence GNU (LGPL) en open source avec services Corba, outils d'administration, connecteurs JDBC.	www.dbxml.org
Software AG	Tamino XML Server	Conçu dès l'origine pour stocker nativement des documents XML, Tamino est la première base de ce type à arriver sur le marché. Il gère les documents XML et assure l'interrogation par le langage de requête XML Query qui est inspiré du XQUERY du W3C, les tendances de Tamino sont les services Web (UDDI, SOAP ...)	www.softwareag.com
X-Hive Corporation	Xhive	X-Hive est un DOM persistant, et il construit à base des standards W3C (XPath 1.0, XPointer, XLink, DOM, XSL-T et XSL-FO), il supporte aussi le langage d'interrogation Xquery et de mise à jour XUpdate.	www.x-hive.com
eXist	eXist-db	eXist est une base de données Open Source native XML, entièrement écrite en Java. permet de formuler des requêtes par XPATH et XQUERY. Elle fournit également des extensions à XPath, comme des fonctions adaptées à la recherche textuelle et aux concepts plus orientés bases de données. Le langage de requête de mise à jour XUPDATE est également supporté.	www.exist.sourceforge.net

Tableau 1.3. Principaux SGBD XML natifs

4. Conclusion

Nous avons présenté dans ce chapitre toute technologie qui tourne autour du langage XML, jusqu'à nous avons abouti à la présentation des caractéristiques des bases de données XML et bases de données XML natives. Les SGBD qui gèrent ce type de base de données sont conçus et implémentés pour assurer la persistance des documents et données XML afin de profiter de l'indexation, d'un moteur de requête, le partage, la sécurité, et autres fonctionnalités typique aux SGDB.

Nous avons présenté aussi les modèles de structuration des documents XML : XML Schema et DTD. L'étude de ces deux technologies de structuration nous incite à choisir XML Schema par rapport au (DTD), malgré la simplicité de ce dernier, comme modèle pour la base de données XML native dans notre travail. Ce choix est justifié par le fait qu'avec les DTD il n'est pas possible de définir des contraintes comme le nombre de fois qu'un élément particulier doit apparaître dans le document, le type des données de chaque élément, les DTD ne sont pas au format XML (structure de balisage) et ne supportent pas les espaces de nom.

Le langage de requêtes XQuery pour l'interrogation des documents et base de données XML est défini dans ce chapitre parce que c'est le langage le plus utilisé et standardisé par le W3C. ainsi que il est très proche dans sa syntaxe au SQL. Il sera adopté comme langage de requête syntaxique pour SGBD XML natif dans l'architecture de notre travail.

La richesse des fonctionnalités du langage de transformation XSLT qui permet de transformer un document XML en un autre document XML ou vers d'autres formats à base de balise est évaluée dans ce chapitre. XSLT servira dans notre travail pour concrétiser les mécanismes de transformation.

XML est développé pour traiter syntaxiquement des documents. Bien qu'une manipulation sémantique ne soit pas assurée par XML, il est considéré comme une référence de base et la première pierre pour les langages du Web sémantiques. XML est un langage qui permet de définir d'autres langages, afin de donner du sens aux données, ce qui fait l'objet du deuxième chapitre.

Les SGBD XML représentent un niveau syntaxique de définition et de manipulation des données, sans tenir compte de la sémantique. De ce fait, notre travail consiste à étendre les fonctionnalités d'un SGBD XML natif par une couche représentant un niveau sémantique afin d'avoir une représentation plus expressive et de permettre l'inférence sur les bases de données. Cette couche sémantique est décrite par le langage d'ontologie Web OWL qui sera décrit dans le deuxième chapitre.

Ontologie et bases de données à base ontologique

1. Introduction

Les derniers travaux menés dans le domaine des bases de données visent à augmenter la capacité de représentation de l'information avec des modèles plus expressifs et riches. Depuis le modèle relationnel, les bases déductives, actives, objet, relationnel-objet jusqu'au modèle XML.

En parallèle, les travaux menés en modélisation des connaissances ont fait émerger la notion des langages du Web sémantique [22]. Contrairement au modèle conceptuel qui prescrit les informations qui doivent être représentées dans une base de données pour répondre à un besoin bien déterminé à priori. Un langage du web sémantique vise à décrire de façon consensuelle les informations permettant de conceptualiser des domaines d'application assez large.

Le Web sémantique est une extension du Web actuel, d'un Web de présentation vers un Web dont les informations ont une sémantique bien définie, et qui permet la coopération entre les humains et les machines [22]. L'objectif des langages du Web sémantique, est de fournir des services nécessaires et suffisants, et selon les attentes sans contribution majeure des utilisateurs. Dans cette optique, les langages du Web sémantique doivent avoir des modèles de présentation et de structuration formels, ainsi que des langages de requêtes de recherche et d'extractions sémantiques des informations efficaces, pour que le Web sémantique soit manipulable par les machines.

Le contenu de ce chapitre est le suivant. Nous décrivons d'abord les langages d'assertions et d'annotations RDF[17] et RDFS[18] qui déterminent les relations entre les ressources ou objets, et qui permettent aussi l'annotation des ressources. Puis, nous présentons le langage de présentation des ontologies OWL[1] et ses capacités de présentation des classes et leurs propriétés avec d'autres constructeurs, ensuite dans le même contexte nous donnons un aperçu sur le langage de requête sémantique SPARQL[14]. Enfin nous définissons les bases de données à base ontologique qui se situent dans l'intersection de deux domaines : les bases de

données et les ontologies. En outre, une synthèse avec commentaires de quelques travaux dans le domaine des bases de données à base ontologique est fournie dans ce chapitre.

2. Langages d'assertions et d'annotations

Les langages d'assertions et d'annotations permettent de donner un cadre sémantique[22] aux documents. D'une part, par la détermination des relations entre les objets, et d'autre part par l'utilisation des méta données[2], tout en permettant l'exploitation des documents par les agents humains et par les agents logiciels, y compris, le mécanisme d'inférence.

2.1. RDF (Resource Description Framework)

XML fournit une syntaxe pour coder et structurer des données, ainsi qu'il a rendu possible la création de nouveaux langages Web pour rendre le partage et la manipulation des connaissances possibles. C'est dans cet esprit qu'a été créé, en 1999 RDF [17], un langage RDF permettant de décrire des métadonnées et facilitant leur traitement automatique ; Il permet aussi l'action de partage et d'échange de connaissances, ainsi que la coopération entre les applications [15].

Le modèle de base de RDF comporte quatre types d'objets:

1. Les **ressources** sont toutes les choses décrites par des expressions de RDF. Une ressource peut varier d'une page entière de Web à un élément d'un document XML. Chaque ressource a un identifiant unique (URI[3][31][32]: Uniform Ressource Identifier).

2. Une **propriété** est un aspect, une caractéristique, un attribut, ou une relation spécifique employée pour décrire une ressource.

3. La notion de **literal** est employée pour identifier des valeurs telles que des nombres et des dates au moyen d'une représentation lexicographique.

4. Un **statement** se compose d'une ressource spécifique ainsi que d'une propriété plus la valeur de cette propriété pour cette ressource. Chaque statement est représenté par un triplet qui se compose d'un sujet (une ressource), d'un prédicat (une propriété) et d'un objet (ressource ou un literal). Un ensemble de tels triplets s'appelle un graphe de RDF, comme il est illustré

[2] Ressource qui fournit des informations sur elle même.
[3] Courte chaîne de caractères identifiant une ressource Web physique ou abstraite (documents, images, élément d'un document, services…) d'une manière unique, et dont la syntaxe respecte une norme d'Internet mise en place pour le World Wide Web.

dans la figure (fig 2.1.). Dans le graphe, chaque triplet représente l'existence d'une relation entre les ressources symbolisées par les nœuds qui sont joints.

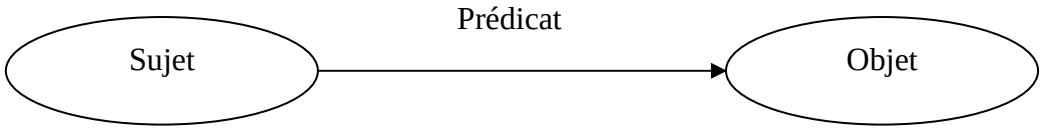


Fig 2.1. Graphe RDF

Les éléments de ces triplets peuvent être des URIs (Universal Resource Identifiers), des littéraux ou des variables.

L'exemple la figure (fig 2.2) détermine que le courrier numéro 14582 est envoyé par la wilaya de Constantine, identifiée par le code W25, vers la ministère de l'intérieur et le cabinet de la wilaya de Constantine, et dont l'objet est une visite présidentielle. Cet exemple est illustré par un graphe RDF dans la figure (fig2.3.):

Le fichier courrier courrier.RDF

```
< ? xml  version= ‘’1.0’’  encoding="UTF-8"  ?>
<rdf:RDF  xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
xmlns:cr="http://www.ministère-intérieure.dz/courrier#">
<rdf:Description rdf:about="courrier_14582">
        <rdf:type rdf:resource="#cr:courrier"/>
        <cr:numéro_courrier>14582</cr: numéro_courrier>
        <cr:objet>Visite présidentielle</cr:objet>
        <cr:contenu>A l'occasion de Youm ElIlm une visite présidentielle est prévue le 16
                avril …….. </cr:contenu>
        <cr:envoyé_par rdf:resource="W25"/ >
        <cr:envoyé_à>
                <rdf:Seq>
                        <rdf:li   rdf:resource="M16"/ >
                        <rdf:li   rdf:resource="C25"/ >
                </rdf:Seq>
        </cr:envoyé_à>
</rdf:Description>
<rdf:Description rdf:about="W25">
        <cr:nom_service>Wilaya Constantine</cr: nom_service>
</rdf:Description>
<rdf:Description rdf:about="M16">
        <cr: nom_service>Ministère de l'intérieur</cr: nom_service>
</rdf:Description>
<rdf:Description rdf:about="C25">
        <cr: nom_service>Cabinet Wilaya Constantine</cr: nom_service>
</rdf:Description>
</rdf:RDF>
```

Fig 2.2. Exemple fichier RDF

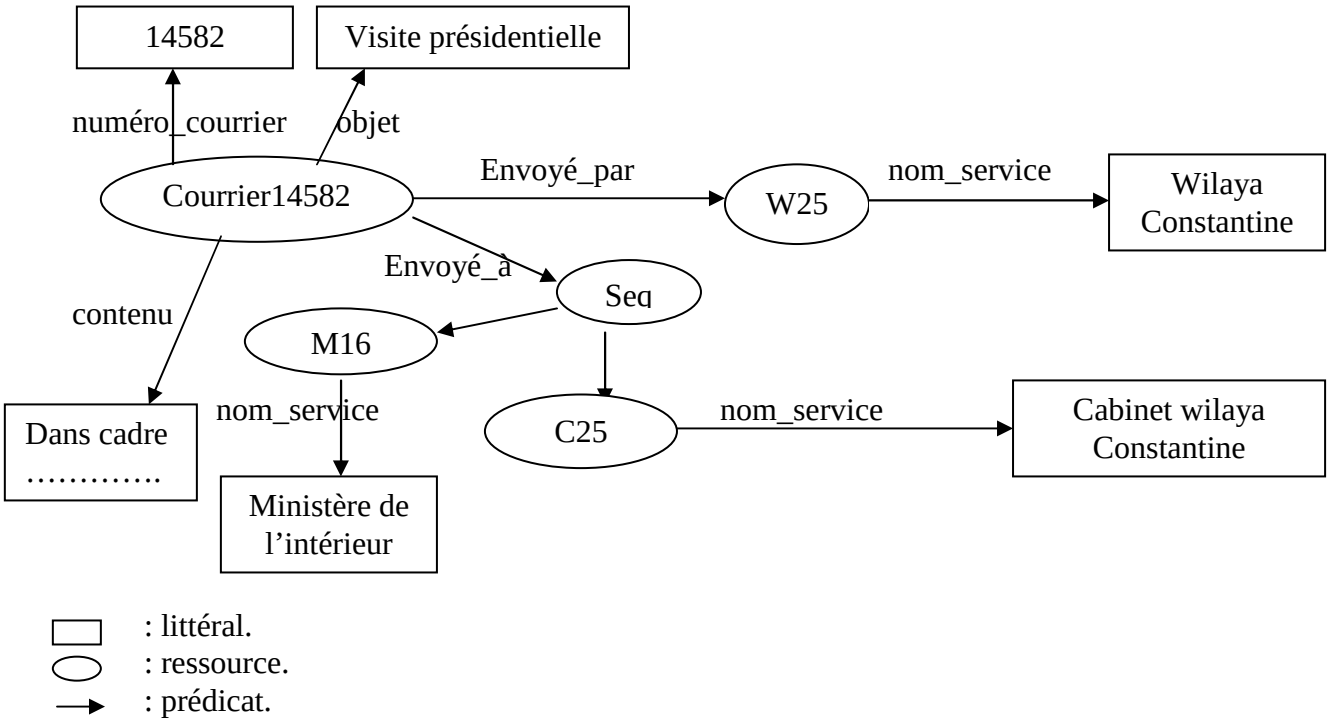


Fig 2.3. Graphe RDF de l'exemple courrier.rdf

RDF propose aussi certains mots-clés réservés, qui permettent de donner une sémantique particulière à des ressources. Ainsi, on peut représenter des ensembles d'objets par les conteneurs: **rdf:bag**: (conteneur non ordonné), **rdf:Seq**: (conteneur ordonné), **rdf:Alt**: (ensemble d'alternative).

2.2. RDFS (Resource Description Framework Schema)

Il est nécessaire, pour donner une sémantique ou un sens aux informations stockées sous forme de triplets RDF, de se donner un vocabulaire (classes et propriétés). RDFS [18] est un langage extensible de représentation des connaissances qui permet de décrire précisément par vocabulaires les ressources d'un domaine donné et les relations entre elles. Ce vocabulaire est utilisé pour annoter les ressources en RDF comme il est illustré dans la figure (fig 2.4).

Les éléments de base du RDFS sont:

- ✓ rdfs:Class : la classe de toutes les classes.
- ✓ rdf:Property : la classe de toutes les propriétés.
- ✓ rdfs:Ressource : la classe de toutes les ressources.
- ✓ rdfs:Literal : la classe des valeurs littérales (chaînes caractères,entiers…).
- ✓ rdf:Statement : la classe des déclarations RDF.

- ✓ rdf:type : permet d'associer une ressource à sa classe (relation classe-instance).
- ✓ rdfs:subClassOf : permet d'associer une classe à l'une de ses super-classes.
- ✓ rdfs:subPropertyOf : permet d'associer une propriété à l'une de ses super-propriétés.
- ✓ rdfs:domain : C'est d'où la relation part. Il permet de spécifier le domaine d'une Propriété, la classe de toutes les ressources qui peuvent apparaître comme sujet dans un triplet Sujet-Propriété-Valeur.
- ✓ rdfs:range : C'est d'où la relation arrive. Cet élément spécifie la classe de toutes les ressources qui peuvent apparaître comme valeur d'une propriété dans un triplet Sujet-Propriété-Valeur.

L'exemple la figure (fig2.4) détermine le schéma associé au graphe RDF de la figure (fig 2.3). Cet exemple est illustré par la figure (fig2.5.):

Le schéma du fichier courrier schéma.RDF

```
<rdf:RDF>
<rdfs:Class rdf:ID='Service'/>
<rdfs:Class rdf:ID='Courrier'/>
<rdfs:Class rdf:ID='Emetteur'>
        <rdfs:subClassOf rdf:resource='#Service'/>
</rdfs:Class>
<rdfs:Class rdf:ID='Destinataires'>
        <rdfs:subClassOf rdf:resource='#Service'/>
</rdfs:Class>
<rdf:Property rdf:ID=' Envoyé_à'>
        <rdfs:domain rdf:resource='#Courrier'/>
        <rdfs:range rdf:resource='#Destinataires'/>
</rdf:Property>
<rdf:Property rdf:ID=' Envoyé_par'>
        <rdfs:domain rdf:resource='#Courrier'/>
        <rdfs:range rdf:resource='#Emetteur'/>
</rdf:Property>
<rdf:Property rdf:ID='objet'>
        <rdfs:domain rdf:resource='#Courrier'/>
        <rdfs:range rdf:resource= '&rdfs;Literal'/>
</rdf:Property>
<rdf:Property rdf:ID='contenu'>
        <rdfs:domain rdf:resource='#Courrier'/>
        <rdfs:range rdf:resource= '&rdfs;Literal'/>
</rdf:Property>
<rdf:Property rdf:ID='numéro_courrier'>
        <rdfs:domain rdf:resource='#Personne'/>
```

```
        <rdfs:range rdf:resource= '&rdfs;Integer'/>
</rdf:Property>
<rdf:Property rdf:ID='nom_servive'>
        <rdfs:domain rdf:resource='#Service'/>
        <rdfs:range rdf:resource= '&rdfs;Literal'/>
</rdf:Property>
</rdf:RDF>
```

Fig 2.4. Schéma RDFS associé au courrier.RDF

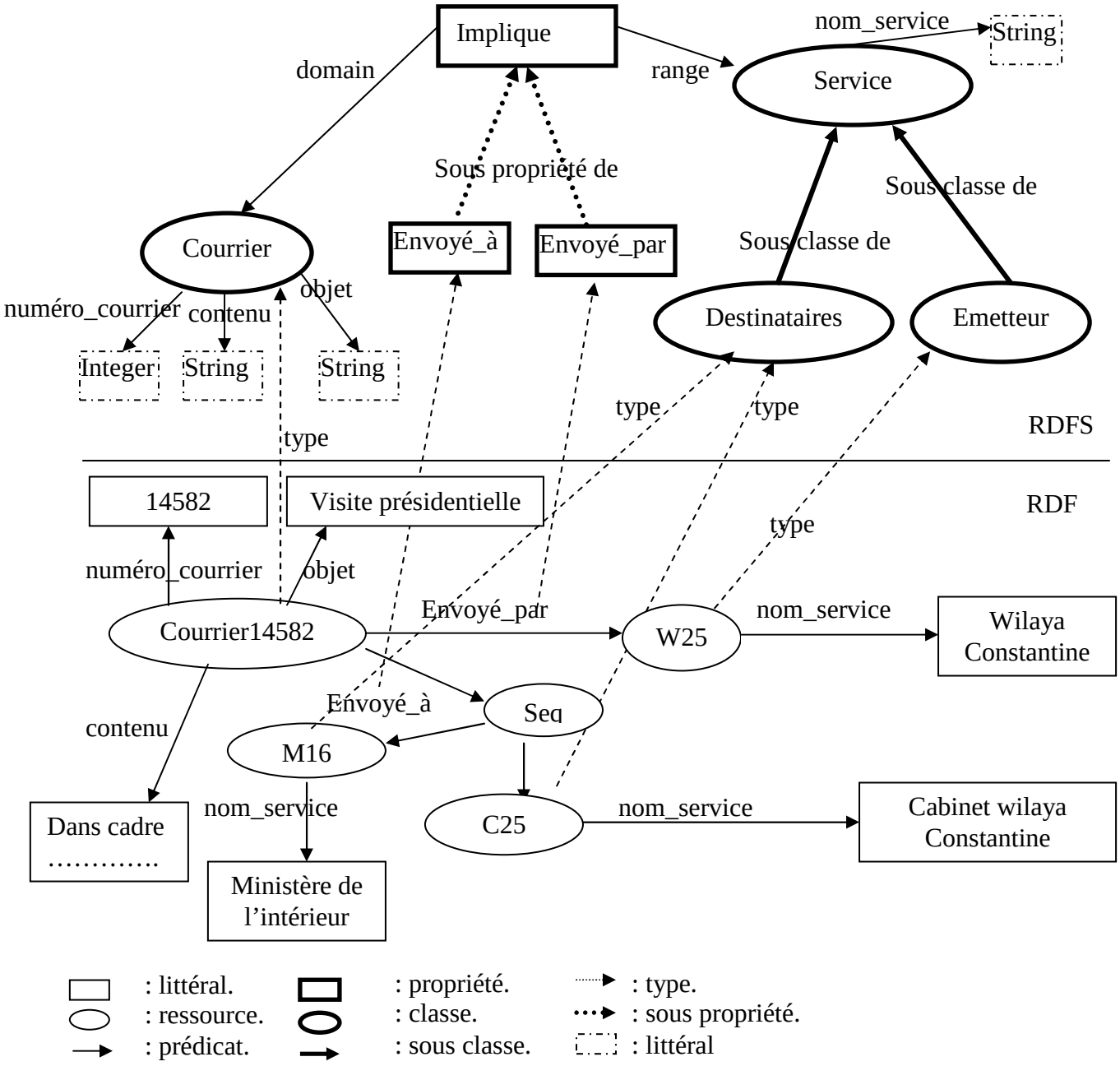


Fig 2.5. Graphe RDF associé à un RDFS

RDFS ne fournit que des mécanismes primitifs pour spécifier ces classes. Il révèle des insuffisances et des limites:

- ✓ RDFS ne permet pas d'exprimer que deux classes sont disjointes. Par exemple, les classes des courriers et des télégraphes sont disjointes.
- ✓ RDFS ne permet pas de créer des classes par combinaison ensembliste d'autres classes (intersection, union, complément).
- ✓ RDFS ne permet pas de définir de restriction sur le nombre d'occurrences de valeurs que peut prendre une propriété. Par exemple, nous ne pouvons pas spécifier qu'un courrier a exactement un seul objet.

Si ces contraintes d'expressivité se montrent trop importantes, nous devons monter à un autre niveau d'expressivité des contraintes sémantiques plus riches du domaine des connaissances.

3. Présentation du langage d'ontologie OWL

Le langage d'ontologie Web OWL [1] est conçu pour décrire et représenter un domaine de connaissance spécifique, en définissant des classes de ressources ou objets et leurs relations ; ainsi que de définir des individus et affirmer des propriétés les concernant et de raisonner sur ces classes et individus dans la mesure où le permet la sémantique formelle du langage OWL. OWL est un standard basé sur la logique de descriptions[19], il est construit sur RDF et RDFS et utilise la syntaxe RDF/XML.

Le langage OWL permet d'étendre les technologies de base (XML, RDF, RDFS) pour apporter :

- ✓ Plus d'interopérabilité (équivalences) ;
- ✓ Plus de raisonnements (logique de description) ;
- ✓ Plus d'évolution (intégration d'ontologies).

Les ontologies OWL se présentent, généralement, sous forme de fichiers texte et de documents OWL.

3.1. Types du langage OWL

Le langage OWL offre trois sous langages d'expression [1][9] conçus pour des communautés de développeurs et d'utilisateurs spécifiques.

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

- Le langage *OWL DL* concerne les utilisateurs souhaitant une expressivité maximum sans sacrifier la complétude de calcul (inférences) et la décidabilité des systèmes de raisonnement. Le langage OWL DL comprend toutes les structures de langage d'OWL avec des restrictions comme la séparation des types (une classe ne peut pas être en même temps un individu ou une propriété, une propriété doit être un individu ou une classe).

- Le langage *OWL Full* concerne les utilisateurs souhaitant une expressivité maximum et la liberté syntaxique de RDF sans garantir le calcul (raisonnement). Par exemple, dans OWL Full, on peut simultanément traiter une classe comme une collection d'individus et comme un individu à part entière. Une autre différence significative par rapport à OWL DL réside dans la possibilité de marquer un objet `owl:DatatypeProperty` comme étant un objet `owl:InverseFunctionalProperty`.

3.2. Eléments du langage OWL

Nous présentons ici des explications exhaustives du rôle de chacun des éléments constituant une ontologie OWL.

3.2.1. En tête OWL

Une fois les espaces de nommage établis, nous introduisons un ensemble d'assertions sur l'ontologie regroupées dans une balise `owl:Ontology[1]`. Ces balises se chargent de tâches communes comme le nom de l'ontologie (rdf :about), les commentaires (rdfs :comment), le contrôle de version (owl:priorVersion) et l'inclusion d'autres ontologies (owl:imports), comme il est détaillé dans l'exemple ci-dessous :

<owl:Ontology rdf:about="Courrier">

<rdfs:comment>Une ontologie OWL de courrier qui servira d'exemple pour tout le mémoire et même pour XML, RDF et RDFS</rdfs:comment>

<owl:priorVersion rdf:resource="20081109/Courrier"/>

<owl:imports rdf:resource="Documents"/>

<rdfs:label>Ontologie sur le trafic du courrier</rdfs:label>

...

</owl:Ontology>

3.2.2. Elément classe

Une classe définit un groupe d'individus possédant des caractéristiques similaires. L'ensemble des individus d'une classe est désigné par le terme extension de classe, chacun de ces individus étant alors une instance de la classe. Chaque individu du monde OWL est un membre de la classe `owl:Thing`. Chaque classe définie par l'utilisateur est donc implicitement une sous-classe de `owl:Thing` [1][9].

La déclaration d'une classe owl se fait de six manières [1][9] :

1-Nommer directement les classes : <owl:Class rdf:ID="Courrier"/>

Nous utilisons `rdf:ID=` "Courrier" pour introduire un nom, comme partie de sa définition. Nous pouvons maintenant appeler la classe Courrier au sein du document par `#Courrier`, par exemple `rdf:resource='#Courrier'`.

Le concept d'héritage existe en OWL à l'aide de la propriété subClassOf. Il existe dans toute ontologie OWL une superclasse, nommée Thing, dont toutes les autres classes sont des sous-classes. Il existe une classe nommée noThing, qui est sous-classe de toutes les classes OWL. Cette classe ne peut avoir aucune instance. Nous définissons, ci-dessous, la classe Courrier comme étant une sous-classe de la classe Document.

<owl:Class rdf:ID="Courrier">

<rdfs:subClassOf rdf:resource="#Document" />

</owl:Class>

2-Enumérer les individus (instances) de la classe :

<owl:Class rdf:ID="Type_destinataires">

<owl:oneOf rdf:parseType="Collection">

<owl:Thing rdf:about="#A"/>

<owl:Thing rdf:about="#Cc"/>

</owl:oneOf></owl:Class>

3-Restreindre les propriétés (valeur et cardinalité) : Une contrainte de valeur s'applique sur la valeur d'une certaine propriété de l'individu, et une contrainte de cardinalité porte sur le nombre de valeurs qui peut prendre une propriété. La propriété cardinalité permet d'exprimer exactement le nombre d'éléments dans une relation ou son intervalle (par exemple, pour un individu de la classe service, Anom_service est une propriété qui est obligatoire. La contrainte de cardinalité portant sur Anom_service restreindra donc la classe décrite aux individus pour lesquels cette propriété apparaît exactement une fois, comme il est indiqué dans l'exemple ci-dessous).

```
<owl:Class rdf:ID="service">
  <rdfs:subClassOf ><owl:Restriction>
   <owl:OnProperty  rdf:resource="#Anom_service"/>
   <owl:cardinality rdf:datatype="&xsd:int">1 <rdfs:cardinality />
  <owl:Restriction/><rdfs:subClassOf />
  <rdfs:subClassOf ><owl:Restriction>
   <owl:OnProperty  rdf:resource="#Acode_service"/>
   <owl:cardinality rdf:datatype="&xsd:int">1<rdfs:cardinality />
  <owl:Restriction/><rdfs:subClassOf />
<owl:Class/>
```

4-Déclarer une nouvelle classe par intersection de deux ou plusieurs classes :

```
<owl:Class>
        <owl:intersectionOf rdf:parseType="Collection">
        <owl:Class rdf:about="#classe1"/>
        …..<owl:Class rdf:about="#classen"/>
</owl:intersectionOf>
</owl:Class>
```

5-Déclarer une classe par union de deux ou plusieurs classes :

```
<owl:Class rdf:ID="classe">
    <owl:unionOf rdf:parseType="Collection">
      <owl:Class rdf:about="#classe1" />
      ….<owl:Class rdf:about="#classen" />
  </owl:unionOf>
</owl:Class>
```

6-Déclarer une classe qui contient tous les individus dans le domaine de discours n'appartenant pas à une autre classe.

```
<owl:Class rdf:ID="classe" />
<owl:Class rdf:ID="Nonclasse">
<owl:complementOf rdf:resource="#classe"/>
</owl:Class>
```

3.2.3. Individus d'une classe

Il suffit de déclarer l'appartenance d'un individu (instance) comme membre d'une classe pour l'introduire[1][9]. Un individu s'exprime de la manière suivante :

<Courrier rdf:ID="Courrier14582">

L'exemple suivant est identique au précédent du point de vue de la signification :

```
<owl:Thing rdf:ID=" Courrier14582" />
<owl:Thing rdf:about="#Courrier14582">
  <rdf:type rdf:resource="#Courrier"/>
</owl:Thing>
```

L'élément `rdf:type` est une propriété RDF liant un individu à une classe dont il est membre.

Une difficulté peut apparaître dans le nommage des individus concerne la non-unicité éventuelle des noms attribués aux individus. Par exemple, un même individu pourrait être désigné de plusieurs façons différentes. C'est la raison pour laquelle OWL propose un

mécanisme permettant de lever cette ambigüité, à l'aide des propriétés : owl:sameAs, owl:diffrentFrom et owl:allDifferent[1].

3.2.4. Elément propriété OWL

Les propriétés nous permettent d'affirmer des faits généraux sur les membres des classes et des faits particuliers sur les individus. Une propriété est une relation binaire. On distingue deux types de propriété [1][9] :

- Les propriétés d'objets (owl:ObjectProperty), qui sont des relations entre les instances de deux classes.
- Les propriétés de types de données (owl:DatatypeProperty), qui sont des relations entre des instances de classes, des littéraux RDF [29] et des types de donnée du schéma XML[12].

a) Propriétés d'objets

Il existe plusieurs façons de restreindre la relation, lorsque nous définissons une propriété. Nous pouvons définir un domaine et une image. Nous pouvons définir la propriété comme une spécialisation (sous-propriété) d'une propriété existante. Des restrictions plus élaborées sont possibles [1][9], comme il est formulé dans l'exemple ci-dessous :

<owl:ObjectProperty rdf:ID="EnvoyéPar">

<rdfs:domain rdf:resource="#courrier"/>

<rdfs:range rdf:resource="#emetteur"/>

<rdfs:subPropertyOf rdf:resource="#implique"/>

</owl:ObjectProperty>

Dans OWL, une séquence d'éléments sans opérateur explicite représente une conjonction implicite. La propriété "EnvoyéPar" a pour domaine la classe "courrier" et pour image la classe "emetteur". C'est-à-dire qu'elle relie des instances de la classe "courrier" à des instances de la classe "emetteur". Plusieurs domaines signifient que le domaine de la propriété est constitué par l'intersection des classes identifiées (idem pour les images).

b) Propriétés de types de données

Dans le cas d'une propriété de type de donnée, l'image de la propriété est un type de donnée. C'est à dire l'image est un ensemble d'individus possédant des types de données. Les

propriétés de types de données recouvrent des littéraux RDF ou des types simples définis conformément aux types de données du schéma XML. Le langage OWL utilise la plupart des types de données intégrés du schéma XML [1][9][12].

Les types de données intégrés du schéma XML plus le type `rdfs:Literal` constituent les types de données intégrés du langage OWL [1].

L'exemple ci-dessous montre que la propriété objet relie les éléments de la classe "courrier" à des valeurs de type chaine de caractère :

<owl:DatatypeProperty rdf.id="objet">

<rdfs:domain rdf:resource="#courrier" />

<rdfs:range rdf:resource="xsd:string"/>

</owl:DatatypeProperty>

c) Propriétés des individus

Dans l'exemple ci-dessous, nous décrivons d'abord les individus de la classe "courrier" puis nous définissons leurs objets :

<courrier rdf:ID=" Courrier14582">

<objet rdf:resource="#visite présidentielle" />

<numéro_courrier rdf:resource="#14582"/>

……..

</courrier>

d) Caractéristiques de propriété

Il existe d'autres moyens pour lier des mécanismes utilisables pour définir des propriétés supplémentaires. Il est possible d'ajouter des caractéristiques de propriété, qui fournissent un mécanisme puissant pour améliorer le raisonnement lié à cette propriété.

Parmi les caractéristiques de propriétés principales, nous trouvons la transitivité, la symétrie, la fonctionnalité et l'inverse [1]. Comme il est détaillé dans le tableau (2.1).

Propriété	Sémantique de la propriété	Exemple
owl:TransitiveProperty	Une propriété P est dite transitive si pour tout x, y, z, P(x,y) et P(y,z) implique P(x,z).	<owl:ObjectProperty rdf:ID="localisé-à"> <rdf:type rdf:resource="owl:TransitiveProperty" /> <rdfs:domain rdf:resource="owl:Thing" /> <rdfs:range rdf:resource="#Région" /> </owl:ObjectProperty> < Région rdf:ID="Massinissa"> <locaklisé rdf:resource="#Elkhourb" /> </Region> <Région rdf:ID="Elkhroub"> <locakisé-à rdf:resource="#Consatntine" /> </Region>
owl:SymmetricProperty	Une propriété P est dite symtrique si pour tout x, y P(x,y) implique P(y,x).	<owl:ObjectProperty rdf:ID="ami"> <rdf:type rdf:resource="owl:SymmetricProperty" /> <rdfs:domain rdf:resource="#Algérien" /> <rdfs:range rdf:resource="#Tunisien" /> </owl:ObjectProperty>
owl:FunctionalProperty	Si une propriété P est considérée fonctionnelle, alors pour tout x, y, z, P(x,y) et P(x,z) implique y = z.	<owl:ObjectProperty rdf:ID=" nombre_courrier_envoyé "> <rdf:type rdf:resource="owl:FunctionalProperty" /> <rdfs:domain rdf:resource="#courrier" /> <rdfs:range rdf:resource="#Valeur" /> </owl:ObjectProperty>
owl:inverseOf	Une propriété P1 est dite inverse du P2 si pour tout x, y P1(x, y) équivalent P2(y, x)	</owl:ObjectProperty> <owl:ObjectProperty rdf:ID="envoyé-à"> <owl:inverseOf rdf:resource="# envoyé-par" /> </owl:ObjectProperty>
owl:InverseFunctionalProperty	Si une propriété P est considérée inverse fonc-tionnelle, alors pour tout x, y, z, P(y,x) et P(z,x) implique y = z	<owl:ObjectProperty rdf:ID=" nombre_courrier_envoyé " /> <owl:ObjectProperty rdf:ID=" envoyé-à "> <rdf:type rdf:resource= "&owl;InverseFunctionalProperty"/> <owl:inverseOf rdf:resource="# envoyé-par " /> </owl:ObjectProperty>

Tableau 2.1. Caractéristiques de propriété

3.2.5. Restrictions de propriété

Il est possible de contraindre de plusieurs façons l'image d'une propriété en fonction de contextes particuliers [1]. Nous le faisons avec des restrictions de propriété (voir tableau (2.2)) ci-dessous. Nous ne pouvons employer les diverses formes décrites que dans le contexte d'un élément `owl:Restriction`. L'élément `owl:onProperty` désigne la propriété restreinte.

Restriction de propriété	Explication	Exemple
owl: allValuesFrom	Exprimer une restriction universelle	<owl:Class rdf:about="#Cours_pg"> <rdfs:subClassOf rdf:resource="#Cours"/> <rdfs:subClassOf> <owl:Restriction> <owl:onProperty rdf:resource="#enseigné-par"/> <owl:allValuesFrom rdf:resource="#Proffesseur"/> </owl:Restriction> </rdfs:subClassOf> </owl:Class>
owl: someValuesFrom	Exprimer une restriction existentielle	<owl:Class rdf:about="#Enseignant"> <rdfs:subClassOf> <owl:Restriction> <owl:onProperty rdf:resource="#Enseigne"/> <owl:someValuesFrom rdf:resource="#Cours_1ere_année"/> </owl:Restriction> </rdfs:subClassOf> </owl:Class>
owl: minCardinality owl: maxCardinality owl: Cardinality	Permet d'exprimer exactement le nombre d'éléments dans une relation ou son intervalle.	<owl:Class rdf:about="#cours_pg"> <rdfs:subClassOf><owl:Restriction> <owl:onProperty rdf:resource="#Enseigné-par"/> <owl:minCardinality rdf:datatype="xsd:nonNegativeInteger">1 </owl:minCardinality> </owl:Restriction></rdfs:subClassOf> <rdfs:subClassOf><owl:Restriction> <owl:onProperty rdf:resource="#enseigné-par"/> <owl:maxCardinality rdf:datatype="xsd:nonNegativeInteger">2 </owl:maxCardinality> </owl:Restriction></rdfs:subClassOf></owl:Class>
owl: hasValue	permet de définir des classes selon l'existence de valeurs de propriétés particulières. elle impose une valeur spécifique à toutes les instances de la classe en question.	<owl:Class rdf:about="#Cours"> <rdfs:subClassOf> <owl:Restriction> <owl:onProperty rdf:resource="#enseigné-par"/> <owl:hasValue rdf:resource="#BOURBIA"/> </owl:Restriction> </rdfs:subClassOf> </owl:Class>

Tableau 2.2. Restrictions de propriété

3.2.6. Equivalence entre classes, propriétés et individus

Pour lier un ensemble d'ontologies dans une nouvelle ontologie, il se révèle souvent utile de pouvoir indiquer qu'une classe (ou une propriété) particulière dans une ontologie est équivalente à une classe (ou une propriété) dans une autre.

La propriété `owl:equivalentClass`[1][9] sert à indiquer que deux classes ont précisément les mêmes instances. Remarquez que dans OWL DL les classes représentent simplement des ensembles d'individus et qu'elles ne sont pas elles-mêmes des individus. Par contre, dans OWL Full, nous pouvons employer la propriété `owl:sameAs` entre deux classes pour indiquer qu'elles sont identiques.

De la même façon, nous utilisons `owl:equivalentProperty` pour lier des propriétés.

Le mécanisme d'identité des individus est similaire à celui pour les classes, mais déclare deux individus identiques [1][9]. Voici un exemple :

```
<Cours rdf:ID="BD">
  <owl:sameAs rdf:resource="#Base de données" />
</Cours>
```

Le contraire de la relation sameAs est la relation differentFrom.

Egalement, au contraire de la propriété `owl:equivalentClass`, nous pouvons définir le caractère disjoint d'un ensemble de classes au moyen du constructeur `owl:disjointWith`[1,9]. Il garantit que l'individu membre d'une classe ne puisse pas être simultanément l'instance d'une autre classe définie. Voici un exemple :

```
<owl:Class rdf:ID="Etudiant">
  <rdfs:subClassOf rdf:resource="#Ressources Humaines"/>
  <owl:disjointWith rdf:resource="#Enseignant"/>
  <owl:disjointWith rdf:resource="#Administrateur"/>
</owl:Class>
```

4. Langage de requêtes sémantiques SPARQL

Les langages de requêtes des ontologies peuvent être classés en deux catégories [15] :

a) langages de requête basés sur RDF, comme RDQL[4], SeRQL[5] et la recommandation du W3C SPARQL[14]. Ces langages sont basés sur la notion des triplets RDF[14] et leur sémantique sur la correspondance entre les triplets et les graphes RDF.

b) langages de requêtes basés sur la logique de description, comme SAIQL[6] et SPARQL-DL[15]. Ces langages projettent à offrir un langage de requêtes puissant et expressif combinant les TBOX, RBOX et ABOX pour extraire des connaissances à partir du langage OWL-DL.

Tous les langages de requêtes des ontologies manipulant les triplets RDF se ressemblent. Et nous nous intéressons en particulier au langage SPARQL dans notre mémoire.

SPARQL[14][15][30] est un langage de requêtes pour l'interrogation de métadonnées et l'extraction des données sous forme d'un graphe RDF ou plus exactement un langage d'interrogation de triplets RDF. Un triplet RDF [14] est une association: **{sujet, prédicat, objet }**.

La syntaxe du SPARQL est inspirée du SQL. Il permet d'exprimer deus types de requêtes :

1. Interrogative : extraction par SELECT des sous graphes, à partir du graphe RDF, satisfaisant les conditions spécifiées dans la clause WHERE.
2. Constructive : génération d'un autre graphe à partir du graphe résultat de la requête d'interrogation.

4.1. Syntaxe du SPARQL

Le vocabulaire de graphe RDF est un ensemble de trois sous ensembles disjointes, contenant les termes suivants [14][15] :

1- Un ensemble des URI Vuri.

2- Un ensemble d'identificateurs des nœuds Vbnoeud.

3- Un ensemble des littéraux Vlit.

[4] http://www.w3.org/Submission/2004/SUBM-RDQL-20040109/.
[5] http://www.nlnet.nl/project/sesame/20030529-serql.html.
[6] http://www.uni-koblenz.de/FB4/Institutes/IFI/AGStaab/Research/SAIQL.

Un graphe RDF est un ensemble fini des triplets, tel qu'un triplet RDF est un tuple {sujet, prédicat, objet} exprimé comme suit :

(s,p,o) **(**Vuri U Vbnoeud) x Vuri x **(**Vuri U Vbnoeud U Vlit).

Pour faciliter la définition d'un graphe RDF, nous pouvons le considérer comme une table relationnelle de trois colonne : s : ?sujet, p :?prédicat, o :?objet. Chaque enregistrement ou ligne de cette table est un triplet [30].

Le corp de la requête SPARQL est basé sur les motifs de triplet appelés motifs de graphe élémentaire, tel que chaque sujet, prédicat et objet peut être des variables (Vvar).

La syntaxe des requêtes SPARQL basées sur le motif de graphe est composée généralement de :

1. La clause SELECT : les variables qui doivent apparaître dans les résultats.
2. La clause WHERE : les motifs de graphe élémentaire.
3. La clause OPTIONAL : les motifs du graphe élémentaire optionnels.
4. La clause FILTER : restriction de résultat par des expressions régulières (test).
5. La clause ORDER BY : ordonner le résultat d'une manière ascendante ou descendante.

Par conséquent, une requête SPARQL suit le modèle suivant :

Requête=(MG, ED, TS, R), tel que :

MG : motifs de graphe élémentaires apparaissent dans la clause WHERE.

ED : ensemble de données ou base de connaissance exprimée par un graphe RDF.

TS : transformateur de solution obtenue lors de l'exécution par les constructeurs : DISTINCT, PROJECTION, ORDER, LIMIT, OFFSET.

R : format de retour de résultat exigé par : SELECT, ASK, CONSTRUCT, DESCRIBE.

a) Exemple d'une requête simple:

L'exemple ci-dessous montre une interrogation SPARQL pour trouver tous les courriers et leurs destinataires dans le graphe de données illustré par la figure (fig 2.5).

Le nom des variables est précédé toujours par « ? ».

SELECT DISTINCT ?x ?y

```
WHERE
  { ?x  rdf :type courrier
    ?y   rdf:type destinataires }
ORDER BY ?date_courrier
```

Résultat d'interrogation :

Courrier	Destinataires
Courrier14582	W25

Chaque solution représente une façon de lier les variables sélectionnées aux termes RDF pour que le motif d'interrogation corresponde aux données. L'ensemble de résultats donne toutes les solutions possibles.

DISTINCT est introduit dans cette requête pour éliminer les solutions en double pour ne pas retourner plusieurs fois les mêmes valeurs.

ORDER BY est introduit dans cette requête pour ordonner les résultats obtenus résultat d'une manière ascendante par ASC() ou descendante par DESC().

b) Exemple des filtres:

La clause FILTER du langage SPARQL restreint les solutions à celles que l'expression de filtre est évaluée vrai. L'exemple ci-dessous impose des contraintes sur la variable destinataire sur l'interrogation SPARQL pour trouver tous les courriers et leurs destinataires dans le graphe de données illustré par la figure (fig 2.5).

```
SELECT ?x ?y
WHERE
  { ?x  rdf :type courrier
    ?y   rdf:type destinataires
FILTER (destinataire= « W25») }
```

Résultat d'interrogation :

Courrier	Destinataires
Courrier14582	W25

c) Exemple de OPTIONAL:

Le graphe RDF n'assure pas la présence de structures complètes et régulières pour toutes les données. Il est parfois nécessaire de ne pas rejeter la solution parce que des parties du motif d'interrogation ne correspondent pas. Cette fonctionnalité est assurée par la clause OPTIONAL.

Nous pouvons, aussi, avoir des contraintes (FILTER) dans un motif de graphe optionnel.

L'exemple ci-dessous montre qu'il existe des motifs de graphe élémentaires qui sont optionnelles pour trouver tous les courriers et leurs destinataires et commentaires dans le graphe de données illustré par la figure (fig 2.5).

SELECT ?x ?y ?commentaire

WHERE

{ ?x rdf :type courrier

?y rdf:type destinataires

OPTIONAL { ?courrier c :A ?commentaire}

FILTER (destinataire= « W25 ») }

Résultat d'interrogation :

Courrier	Destinataires	Commentaire
Courrier14582	W25	

d) Exemple de UNION:

Le langage de requête SPARQL assure la combinaison des motifs de graphe, par le mot clé UNION, d'une manière que l'un des motifs de graphe alternatifs peut correspondre.

Chaque alternative de la requête peut contenir plusieurs motifs de triplet.

L'exemple ci-dessous montre comment concaténer les solutions de plusieurs requêtes

SELECT ?title

WHERE {{ ?book dca :title ?title} UNION{ ?book dcb:title ?title }}

Résultat d'interrogation :

Title
Courrier14582

e) Exemple de LIMIT et OFFSET:

OFFSET fait commencer les solutions générées après le nombre indiqué de solutions.

LIMIT met une limite supérieure au nombre de solutions retournées. Si le nombre de solutions réel est supérieur à la limite, alors le nombre limite de solutions, au plus, sera retourné.

L'utilisation de LIMIT et OFFSET, pour sélectionner des sous ensembles différents parmi les solutions d'interrogation, n'aura pas d'utilité sans la clause ORDER BY.

SELECT ?name

WHERE { ?x foaf :name ?name}

ORDER BY ?name

LIMIT 3

OFFSET 5

f) Exemple de ASK:

ASK retourne un booléen indiquant si un motif d'interrogation a une solution ou non.

L'exemple ci-dessous vérifie la correspondance de la variable courrier dans le graphe RDF

ASK { ?x f :name « alice »}

Réponse : No

g) DESCRIBE :

DESCRIBE retourne un graphe RDF décrivant les données de ressources trouvées. Ces données ne sont pas les objets de la clause d'interrogation SELECT.

h) Exemple de CONSTRUCT:

La forme d'interrogation CONSTRUCT retourne un sous graphe RDF à partir du graphe RDF. Ce sous graphe est généré par la construction d'un ensemble des triplets concordant avec le motif de graphe de l'interrogation.

L'exemple ci-dessous montre comment construire un sous graphe à partir du graphe RDF.

CONSTRUCT { ?x c :objet ?objet}

WHERE { ?x courrier :objetcourrier ?objet}

Résultat de construction :

c :courrier c : objet «visite présidentielle »

Le résultat de l'application de CONSTRUCT est un graphe, par contre l'application de SELECT retourne des liaisons de variables sous forme tabulaire.

4.2. Sémantique du SPARQL

Un motif de graphe élémentaire correspond à un sous-graphe des données RDF lorsque les termes RDF de ce sous-graphe peuvent être substitués par les variables et que le résultat est un graphe RDF équivalent au sous-graphe.

Le résultat d'une requête SPARQL appliquée sur un graphe RDF est obtenue suite à la correspondance entre les variables de la requête et les termes du graphe RDF[14][15]. Cette correspondance, qui est pratiquement une substitution des variables, fournit le résultat de la requête qui est un sous-graphe du graphe RDF[14][15][29].

En d'autre terme, si nous considérons que le graphe RDF est une table relationnelle, le résultat de la requête SPARQL est l'ensemble des lignes de la table des triplets satisfaisantes les relations entre objets de la clause WHERE.

5. Base de données à base ontologique

La modélisation des bases de données n'a pas cessé d'évoluer depuis la première génération des systèmes de gestion de base de données basés sur le modèle réseau ou le modèle hiérarchique, vers les années 1970. Les programmes sont dépendants de l'organisation

physique des fichiers sur le disque et les applications doivent être modifiées à chaque réorganisation physique.

Dans les années 1980, la deuxième génération a connu le modèle relationnel qui représente toutes les informations sous forme de tables. La simplicité du modèle a permis la définition de langages de requêtes simples, faciles d'utilisation et puissants (SQL).

Les années 1990 ont vu apparaître de nouvelles applications (CAO, PAO, multimédia, etc.) nécessitant l'utilisation de bases de données. Ces nouvelles applications ont révélé les insuffisances de modèle relationnel. La troisième génération orientés-objet des modèles de base de données sont la réponse directe aux problèmes des nouvelles applications. Les avantages de la programmation par objet sont reconnus : puissance des concepts, programmation modulaire, réutilisation du code et maintenance facile du code.

En 1993, et pour profiter d'une part, du succès commercial et de la simplicité de format physique de données du modèle relationnel, et d'autre part de l'expressivité et l'extensibilité du modèle conceptuel en objet, un autre modèle est apparu qui est le modèle relationnel-objet.

Au fil de l'évolution des modèles de bases de données, des modèles de représentation des connaissances avec des possibilités d'inférences ont connu des évolutions importantes jusqu'à l'arrivée à la technologie à base ontologique OWL proposée par le consortium W3C[1], qui a permis de donner plus de sémantique aux données. Le langage d'ontologie OWL fournit une diversité de services pour modéliser, analyser et rechercher des informations, ainsi que permet de faire des inférences. Néanmoins, si la quantité et la taille des données sont importantes, il n'est pas possible de les manipuler en gérant un fichier simple, il devient nécessaire de les gérer par un système de gestion de base de données qui permet de représenter l'ontologie par une base de données tout en conservant la sémantique et les contraintes, et stocker les données afin de les mieux gérer et les optimiser lors des opérations de recherches et de mises à jours d'une part, et d'autre part les sécuriser et assurer leurs partage. Nous qualifions ces bases de données, des bases de données à base ontologique [33][34].

Le domaine de base de données à base ontologique se situe dans l'intersection de deux domaines : les bases de données et les ontologies. La notion de Bases de Données à Base Ontologique [33][34] possède deux caractéristiques :

1) Ontologie et données sont toutes les deux représentées dans la base de données et peuvent faire l'objet des mêmes traitements (insertion, mise à jour, requêtes, etc.) ;

2) toute donnée est associée à un élément ontologique qui en définit le sens (sémantique).

Etant donné plusieurs bases de données à base ontologique, sont développées séparément en s'appuyant sur la technologie OWL et utilisant la même ontologie de domaine, il sera plus facile d'intégrer les sources de données différentes, et même assurer un entreposage, si la représentation explicite de l'ontologie est au sein des bases de données.

En plus des avantages des ontologies, un Système de Gestion de Base de Données (SGBD) offre des facilités de stockage, d'accès, de manipulation de grands volumes de données. Les SGBD garantissent également la cohérence des données (en cas de mise à jour simultanée par plusieurs utilisateurs), leur intégrité (en cas d'opération incorrecte par un programme ou un utilisateur), leur fiabilité et leur récupération (en cas de panne matérielle ou logicielle) et leur confidentialité (en cas d'accès malveillant ou accidentel).

L'intégration de la sémantique dans les bases des données pourra créer une nouvelle génération des systèmes de gestion des bases de données. C'est dans cette optique, que notre travail s'inscrit.

Notre travail consiste, à établir une conception et une modélisation logique sous le langage OWL, et un stockage physique sous une base de données XML native, en vue d'exploiter pleinement la sémantique offerte par le langage d'ontologie OWL, et les fonctionnalités de gestion et manipulation de données offertes par le système de gestion de base de données XML natif. Notre approche se base sur l'ajout d'une couche sémantique sur le SGBD XML natif, utilisant la technologie OWL-DL au-dessus d'une couche de description et de manipulation syntaxique de données XML.

5.1 Etude des travaux existants

Les derniers travaux menés dans le domaine des bases de données visent à augmenter la capacité de représentation de l'information avec des modèles plus expressifs, riches et permettant l'inférence. Depuis le modèle relationnel, les bases déductives, actives, objet, relationnel-objet jusqu'au modèle XML. Aussi, l'intégration de la sémantique dans le domaine des bases de données est devenue un sujet de recherche ces dernières années.

La modélisation à base d'ontologies permet de décrire à la fois les données et leur sémantique exprimée par des concepts ou des expressions de concepts de l'ontologie. Ainsi, et vu le volume des informations, il est apparu la nécessité de rendre persistantes aussi bien les ontologies que leurs instances. Plusieurs travaux de recherches dans le domaine des bases de données à base ontologique ont été proposés à cet effet, par exemple: OntoDB [33][34], ONTOMS[35], DLDB[36],SESAME[39].

a) Dans le travail [33][34], Hondjack a proposé un nouveau modèle d'architecture de base de données appelé base de données à base ontologique (OntoDB). Il a défini séparément l'implémentation de l'ontologie (Onto) et des données (DB). La solution comporte deux parties. Elle consiste :

   1) A représenter les primitives du modèle d'ontologie au sein d'un méta-schéma (étendre la méta-base native existant dans le SGBD).

   2) A définir, une fois pour toute, le schéma physique de stockage des ontologies.

   Son prototype utilise les ontologies PLIB[7], définies par un modèle EXPRESS[8].

   Son modèle OntoDB est constitué de quatre parties :

   Deux parties traditionnelles des bases de données: méta-base (qui contient les schémas) et données, s'ajoutent, d'une part, l'ontologie qui permet de stocker les différentes ontologies des domaines couverts par la base de données, et, d'autre part, le métaschéma qui contient un méta-modèle réflexif du modèle d'ontologie utilisé, et qui permet de rendre générique tout traitement sur les ontologies. Il a validé cette architecture par un prototype de base de données à base ontologique réalisé sur la base du SGBD PostgreSQL sur une base de données relationnelle objet, et utilisant des ontologies PLIB dans un environnement basé sur EXPRESS.

b) Une architecture appelée Sesame est proposée dans le travail de [39]. Elle est conçue pour le stockage et l'interrogation des données et des métadonnées RDF et RDFS. Sesame est une architecture générique, elle est modélisée et implémentée pour les bases de données : relationnelle, relationnel objet , stockage sous forme triplets. Et ceci sans changer le moteur de requêtes. Sesame utilise conjointement les SGBD MySQL, PostgreSQL, Oracle et SQL server. Le moteur de requête de

[7] PLIB[34][37] permet de définir de façon formelle et automatique toutes les catégories d'objets que l'on peut avoir besoin de manipuler lors d'une transaction sur le Web sémantique, ainsi que les propriétés qui les caractérisent ou décrivent leur état. Il permet ensuite de référencer ces définitions par de simples codes. Comparé, à OWL, le modèle PLIB présente aussi l'avantage de supporter le multi-linguisme, la multi-représentation des concepts et un système de type complet beaucoup plus riche que les simples types chaîne et entier existant en OIL. Il est également bien outillé, référençable par les protocoles de dialogue en cours de développement dans le domaine du commerce électronique et il fait déjà l'objet de premières utilisations industrielles. Le modèle PLIB d'ontologie contient des contraintes d'intégrité sur les objets modélisés mais ne permet pas d'exprimer de règles de raisonnement sur ces objets.

[8] EXPRESS[34] est un langage de modélisation formel de l'information. Il permet à la fois de modéliser les concepts d'un domaine (modèle conceptuel) et de spécifier les structures de données de ces concepts (modèle logique). Son but est de spécifier une base de données. Comme UML, EXPRESS est un langage de modélisation objet mais il possède quelques différences du point de vue de la représentation de certains concepts objets .

Sesame est implémenté par le langage RQL qui a une syntaxe très proche de OQL, et il permet d'établir des requêtes sur les classes, les propriétés et les instances.

c) Un autre travail appelée Jena[9] est proposée par les laboratoires HP. Jena est un Framework constitué d'un ensemble d'outils implémenté en java :

-une API de programmation pour la gestion des données (RDF, RDFS, DAML+OIL, OWL) des applications de Web sémantique.

-un langage de requêtes AQL qui est une implémentation du langage SPARQL.

-une structure relationnelle pour un stockage persistant des données RDF, RDFS, DAML+OIL et OWL.

-un parseur RDF/XML.

-un moteur d'inférence couplé à autres raisonneurs.

-un outil pour migrer les instances d'un format à un autre.

d) Dans le travail de [36], l'architecture DLDB est proposée, c'est une extension par une base de connaissance sémantique DAML+OIL[38] du système de gestion de base de données relationnel avec un mécanisme d'inférence. L'implémentation est faite en utilisant le SGBD relationnel MS Access et le raisonneur de la logique de description FaCT. Un module de requête sémantique est fourni pour interroger l'ensemble des données de l'ontologie DAML+OIL.

e) Le dernier travail présenté dans ce mémoire est l'architecture ONTOMS[35]. C'est un système de gestion de données (instances) OWL, par lequel les données sont stockées physiquement dans des classes modélisées par des tables relationnelles. Ce système assure, en plus du raisonnement canonique, le raisonnement non canonique avec inverseOf, symmetric, sameAs. Il évalue aussi les requêtes exprimées par le langage OWL-QL pour extraire des informations à partir d'un volume important de données des ontologies stockées dans la base de données relationnelle.

5.2 Synthèse des travaux en relation

L'architecture Sesame est conçue pour stocker et interroger des données et des métadonnées RDF et RDFS, elle ne supporte pas les ontologies écrites en OWL. Outre Sesame est modélisée et implémentée pour les bases de données : relationnelle, relationnel objet par un stockage sous forme triplets(sujet, prédicat, objet) sans tenant en compte le

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

modèle des bases de données XML qui conserve une partie de la sémantique lors de la transformation de RDF/S vers DTD ou XML Schema. Le moteur de requête de Sesame RQL n'est pas un standard ou reconnu pas le consortium W3C.

Le Framework Jena proposée par les laboratoires HP est plus expressive en matière de la sémantique par rapport à l'architecture Sesame. Jena supporte aussi le type d'ontologie DAML+OIL et OWL.

Egalement son langage de requêtes AQL qui est une implémentation du langage SPARQL du consortium W3C. Jena comme Sesame est une structure relationnelle pour un stockage persistant des données et ne supporte pas le modèle XML.

L'architecture DLDB est une extension sémantique du SGBD relationnel par DAML+OIL avec un mécanisme d'inférence. Comme Sesame elle ne supporte pas les ontologies écrites en OWL et ne considère pas le modèle XML dans son architecture.

ONTOMS est un système de gestion de données (instances) OWL par des tables relationnelles, il supporte un langage de requête sémantique (OWL-QL) plus évolué par rapport à celui de Sesame et Jena . Il ne considère pas le modèle XML dans son architecture.

L'architecture OntoDB, est une extension du SGBD PostgreSQL par une ontologie de type PLIB, alors c'est facile d'évoluer vers les ontologies OWL, mais comme les architectures précédentes, elle ne considère pas le modèle XML.

6. Conclusion

Dans ce chapitre nous avons présenté les langages d'assertions et d'annotations RDF et RDFS. Puis, le langage de présentation des ontologies le plus riche et le plus présentatif de domaine de connaissance OWL avec ses trois types OWL-lite, OWL-DL et OWL-full, par la description de ces concepts et ces composants(classe, propriété, contrainte, etc). Ensuite nous avons donné un aperçu sur le langage de requête sémantique SPARQL. A la fin de ce chapitre nous avons défini un nouveau concept qui est les bases de données à base ontologique avec une synthèse commentée de quelques travaux proposés dans la littérature.

Les travaux menés en modélisation des connaissances faisaient émerger la notion de l'ontologie OWL ; et les derniers travaux menés dans le domaine des bases de données visant à augmenter la capacité de représentation de l'information avec des modèles plus expressifs et riches ; et la description des principaux travaux dans le domaine des bases de données à base

ontologique nous ont permis de définir les différents composants et concepts utilisés dans notre architecture proposée.

Nous présentons alors dans le chapitre 3, une modélisation sémantique dans les bases de données XML natives par l'extension des fonctionnalités d'un SGBD XML natif par une couche représentant un niveau sémantique exprimée par le langage d'ontologie OWL afin d'avoir une représentation plus expressive et de permettre l'inférence sur les bases de données.

De ce fait, un mécanisme pour transformer une ontologie OWL vers une base de données XML natives en utilisant une description formelle basée sur les schémas XML est détaillé dans le chapitre 3. Ce mécanisme permet d'exécuter les requêtes recherches sémantiques basées sur le langage de requête d'ontologie SPARQL pour manipuler l'ontologie OWL dont les instances (données) sont situées au niveau des nœuds feuilles des documents XML.

Chapitre III

Un modèle architectural sémantique pour les bases de données XML natives

1. Introduction

Les bases de données XML [2] natives représentent un référentiel intéressant pour les documents et données structurés ou semi structurés. Néanmoins les performances des systèmes de gestion des bases de données natifs XML sont actuellement en cours d'amélioration [6][10].

Le processus de recherche et d'extraction des informations pertinentes exige un processus fiable et rapide d'interrogation tout en prenant en charge l'aspect sémantique. Plusieurs langages de requêtes ont été développés pour la recherche et l'interrogation des parties de documents et de données modélisées dans une base de données XML natives. Citons par exemple : XPATH, XQUERY [7][,8]…

Plusieurs langages de représentation de l'information sémantique, plus complexes que XML [2][3], ont été développés afin d'améliorer la recherche sémantique et de raisonner sur les contenus des documents et de données. Le langage OWL (Web Ontology Langage) [1][9] proposé par le consortium W3C. Il, est bâti sur la logique de description et s'appuie sur la syntaxe du langage XML[9]. OWL fournit une diversité de services pour modéliser, analyser et rechercher des informations, ainsi que il permet de faire des inférences [1]. Il fournit aux développeurs plus que XML. Néanmoins, si la quantité et la taille des données sont importantes, il n'est pas possible de les manipuler en gérant un fichier simple, il devient nécessaire de les gérer par un système de gestion de base de données qui permet de représenter l'ontologie tout en conservant la sémantique et les contraintes, et stocker les données afin de les mieux gérer et les optimiser lors des opérations de recherches et de mises à jours d'une part, et d'autre part les sécuriser et assurer leurs partage.

Afin d'exploiter l'aspect sémantique du langage OWL dans le domaine des bases de données, dans ce chapitre, nous allons présenter notre contribution qui est une architecture de

l'extension du SGBD XML natif par une couche sémantique. De ce fait, nous allons délimiter deux axes qui sont:

- Dénombrer les règles de transformation de l'ontologie OWL-DL vers une base de données XML natives en utilisant une description formelle basée sur les schémas XML.
- Faire la correspondance entre les clauses de deux langages de requêtes SPARQL et XQUERY en se servant de l'algèbre relationnelle comme intermédiaire.

2. Approche globale

Le modèle sémantique d'une base de données, proposé, est basé sur la combinaison entre les deux technologies à savoir : le langage d'ontologie Web OWL-DL [1][4][9] et les bases de données XML natives [10][11]. C'est une approche se basant sur l'ajout d'une couche sémantique sur le SGBD XML natif, utilisant la technologie OWL-DL au-dessus d'une couche de description et de manipulation syntaxique de données XML.

L'intersection de deux domaines : les bases de données et les ontologies, entre dans le cadre que nous pouvons le qualifier par la notion de Bases de Données à Base Ontologique.

Dans ce chapitre, nous présentons un aperçu sur les différents modules (composants) et concepts de ce modèle sémantique.

La couche sémantique de cette modélisation est constituée de deux composants :

1. le module des connaissances sous le langage d'ontologie OWL-DL.
2. le module de la prise en charge des requêtes (de recherche) sémantiques sous le langage de requête sémantique SPARQL [14][15].

Les deux composants du niveau sémantique seront transformés automatiquement vers leurs homologues du niveau syntaxique géré par le noyau du SGBD XML natif.

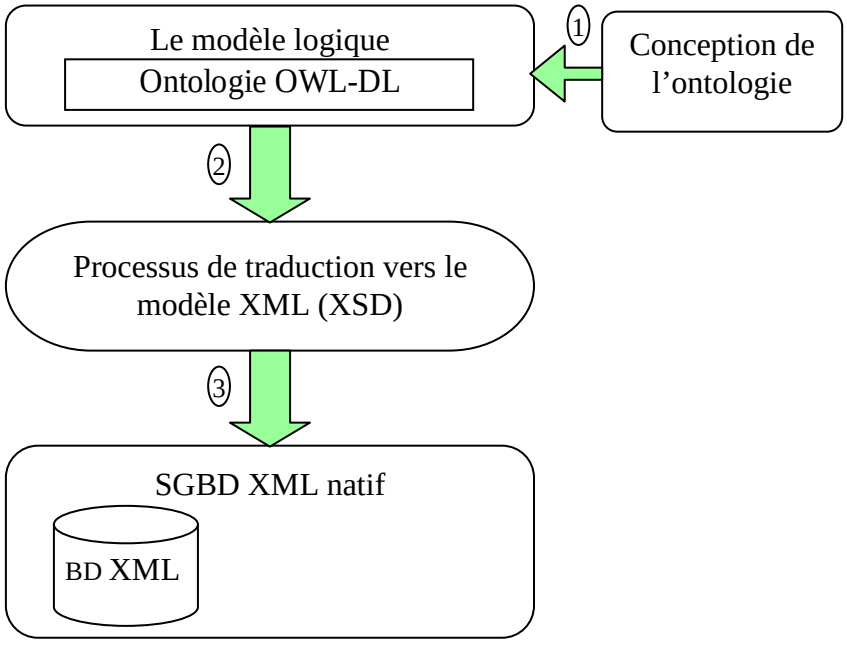
L'ajout de la couche sémantique dans notre modèle proposé a pour but de séparer l'ontologie et les instances (données) :

1. L'ontologie représente les classes de ressources ou objets et leurs relations. c'est avec l'ontologie que l'utilisateur interagit.
2. La donnée est stockée dans la base de données XML native et associée à un élément de schéma XML de la base de données qui est lui même correspond à un élément de l'ontologie (le sens de la donnée est assuré indirectement).

L'objectif est de permettre une conception et une modélisation logique sous le langage OWL, et un stockage physique sous une base de données XML native, en vue d'exploiter pleinement la sémantique offerte par le langage d'ontologie OWL, et les fonctionnalités de gestion et manipulation de données offertes par le SGBD XML natif.

2.1. Module des connaissances sous OWL

Le modèle conceptuel du domaine de connaissances est représenté par le langage d'ontologie OWL-DL, comme il est illustré par la figure (fig 3.1). C'est avec ce type de structure que l'interface utilisateur interagit. Le type de langage d'ontologie OWL-DL est utilisé pour exprimer l'ontologie par rapport aux deux autres types OWL-FULL et OWL-LITE. Il possède une expressivité maximum sans sacrifier la complétude de calcul (inférences) et la décidabilité des systèmes de raisonnement. Le langage OWL DL comprend toutes les structures de langage d'OWL avec des restrictions comme la séparation des types (une classe ne peut pas être en même temps un individu ou une propriété). Aussi il permet d'exprimer des cardinalités supérieures [1].



1 : dés le début, la conception se fait avec la considération que le résultat de la phase conception est une ontologie OWL-DL.

2 : le modèle logique de l'ontologie OWL-DL est traduit vers le schéma XML.

3 : le schéma XML est stocké dans la base de données XML native.

Fig 3.1. Traduction de l'ontologie OWL-DL

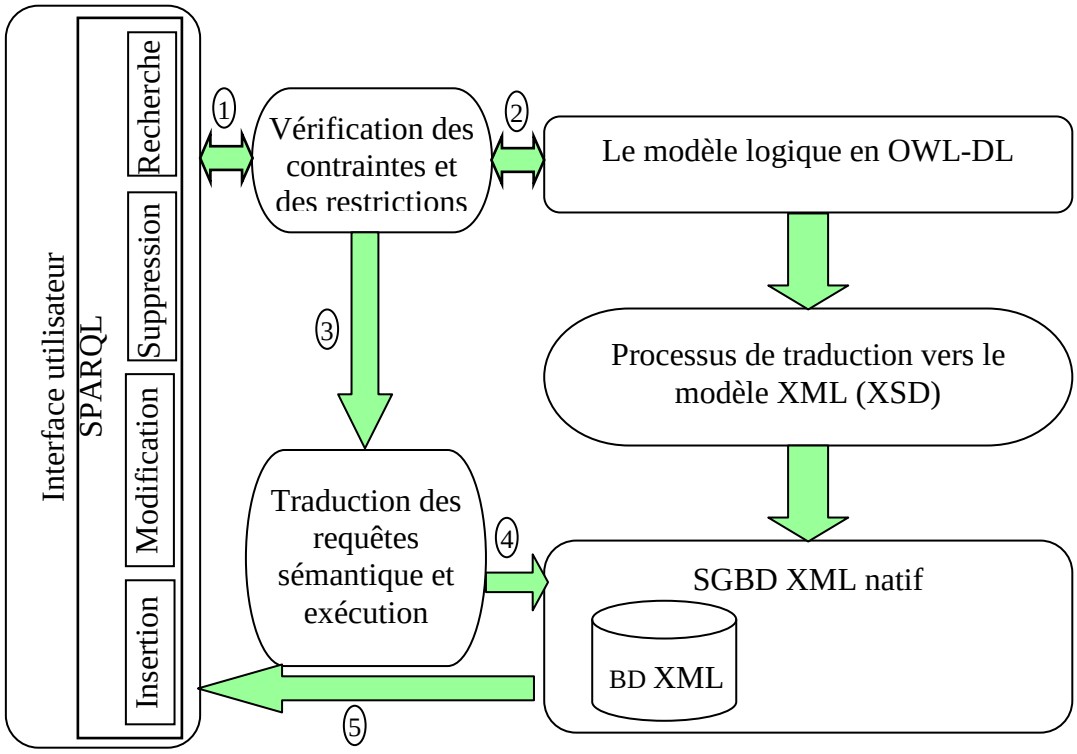
La couche XML utilise comme modèle de structuration la technologie XML Schema [12]. Le choix de XML Schema par rapport au modèle Document Type Definition (DTD) [10][2][3] malgré la simplicité de ce dernier, est justifié par le fait qu'avec les DTD il n'est pas possible de définir des contraintes comme le nombre de fois qu'un élément particulier doit apparaître dans le document, le type des données de chaque élément, les DTD ne sont pas au format XML (structure de balisage) et ne supportent pas les espaces de nom [10].

Le modèle conceptuel et logique est sous la description OWL-DL, et la persistance des instances (données) est assurée par un modèle physique géré par le noyau du SGBD XML natif, comme il est illustré par la figure (Fig 3.1).

Un processus de traduction du modèle sémantique logique, exprimé en OWL-DL, vers la structure syntaxique, exprimée en XML Schema, assure le mécanisme de correspondance entre les éléments de ces deux technologies. Par conséquent, la conception et la modélisation logique sont assurées par l'ontologie OWL-DL, et le stockage physique est assuré par la base de données XML native.

2.2. Module des requêtes sémantiques

Les instances (données) des classes OWL sont chargées aux nœuds de l'arbre XML afin d'assurer leur gestion, tout en respectant le schéma XML obtenu par le processus de transformation du modèle logique en ontologie OWL vers la base de données XML native (Fig 3.2) sous la structure XML Schema. Les instances des classes OWL sont les individus des attributs d'un document XML ou les individus des éléments d'un document XML qui n'ont pas des fils éléments ou attributs.



1 : la requête sémantique doit être passée par la vérification des contraintes sémantiques et des restrictions de cardinalité avant d'être approuvée pour l'exécution.

2 : les contraintes sémantiques et les restrictions de cardinalité sont vérifiées à partir du modèle logique OWL-DL.

3 : si la requête sémantique est approuvée pour l'exécution, elle va être traduite vers une requête syntaxique supportée par le SGBD XML.

4 : l'exécution de la requête syntaxique par le SGBD XML natif.

5 : le revoie du document XML, résultat de l'exécution de requête.

Fig 3.2. Exécution des requêtes sémantiques

Avant de charger des instances avec une requête d'insertion, supportée par le langage de manipulation des instances (données) sous le langage d'ontologie OWL, if faut vérifier d'abord que cette insertion respecte les contraintes et les restrictions de l'ontologie OWL de notre sujet (Fig 3.2). Si l'insertion est valide, un mécanisme transforme la requête vers une requête en langage supportable par le système de gestion de base de données XML natif (XUPDATE) [6][7][8] et puis provoquer son exécution pour insérer les instances dans la base de données XML native. De la même façon nous assurons l'exécution des requêtes de modification, de suppression des instances et de recherche et d'extraction des données (XPATH, XQUERY).

La recherche et l'extraction sémantiques des connaissances sont effectuées par la technologie SPARQL [14][15]. Cette technologie permet d'exploiter des règles inférences pour la recherche d'information. Le traitement de la requête se fait en trois phases :

1. Traduction de la requête sémantique exprimée par le langage SPARQL vers le langage de requête supporté par le SGBD XML natif (XQUERY) ;
2. Exécution de la requête XQUERY ;
3. La construction de résultat XML.

3. Architecture de la couche sémantique

Nous présentons ici l'architecture qui en découle à l'issue de l'ajout de la couche sémantique au dessus du SGBD XML natif. Cette couche contenant : Le schéma de la base de connaissance exprimé par la technologie à base ontologique OWL-DL, et le gestionnaire de requêtes sémantiques qui tourne sur le langage SPARQL. Comme il est illustré par la figure (Fig 3.3), l'interface utilisateur entre en interaction avec le SGBD XML natif à travers la couche sémantique représentant l'ontologie de son domaine de connaissances. Par conséquent, l'exploiteur des connaissances ne se soucie pas de la manière de représentation syntaxique et physique des données du domaine de connaissance, il n'a devant lui qu'un schéma d'ontologie OWL-DL et langage de requêtes SPARQL.

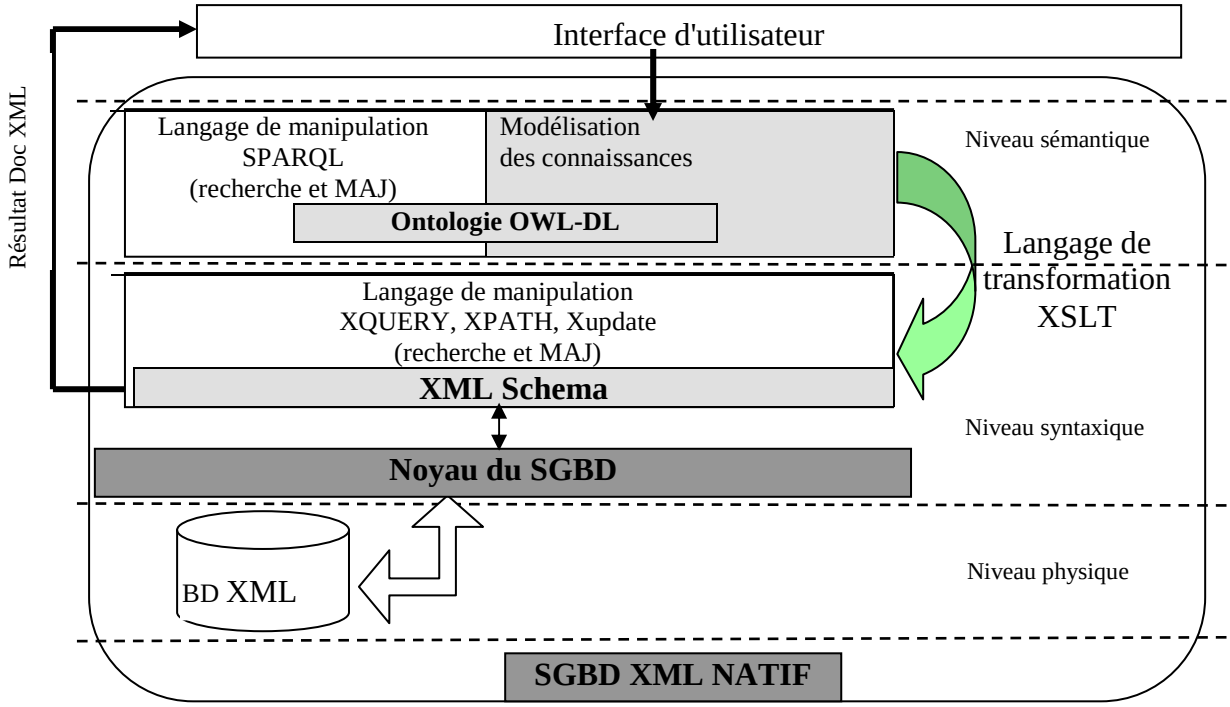


Fig 3.3. Architecture sémantique du SDBG XML natif

Le niveau syntaxique de cette architecture a un lien étroit avec le niveau physique. Ces deux niveaux constituent le SGBD XML natif utilisé dans notre sujet. Le niveau syntaxique inclut le modèle logique de la base de données exprimé par la technologie XML Schema, à l'issue d'une transformation automatique du schéma de la base de connaissance de l'ontologie OWL-DL du niveau sémantique, en utilisant le langage de transformation XSLT[13], comme il est illustré par la figure (Fig 3.3).

Le langage de transformation XSLT est choisi, pour concrétiser le mécanisme de transformation des différents concepts de l'ontologie vers les différents éléments appropriés du schéma XML de la base de données cible, car la syntaxe de la structure du langage OWL est à base XML (structure de balise) [9]. Ceci permettra de localiser les balises sur lesquelles on établit des conditions, et enfin choisir le format cible qui est sous la syntaxe du XML Schema.

Ce sont les requêtes du niveau syntaxique qui vont être réellement exécutées. Elles sont obtenues, à l'instar du modèle de schéma de données, par un mécanisme de transformation des différents composants de la requête sémantique SPARQL vers les différents composants des langages de requête XQUERY et XPATH supportés par le SGBD XML natif, et ceci, après vérification des contraintes sémantiques et de restriction (propriété, cardinalité). Le résultat de l'exécution de la requête sera renvoyé à l'utilisateur sous sa forme native, c'est à dire, un fichier XML, sans transiter par la couche sémantique.

Le niveau physique de cette architecture assure la persistance des instances (données). Le modèle physique géré par le noyau du SGBD XML natif, comme il est illustré par la figure (Fig 3.3).

Si la masse et la taille des connaissances (données) du domaine de l'ontologie OWL-DL sont considérable, il serait complexe de les manipuler en gérant un fichier simple, il devra impératif de les gérer par un système de gestion de base de données qui permet de représenter l'ontologie tout en conservant la sémantique et les contraintes, et stocker les données afin de les mieux gérer et les optimiser lors des opérations de recherches et de mises à jours d'une part, et d'autre part les sécuriser et assurer leurs partage. C'est dans cette optique que l'architecture illustrée par la figure (Fig 3.3) est proposée.

Le choix du modèle de bases de données XML natives par rapport aux autres modèles (relationnel, objet) pour juxtaposer la couche sémantique par le langage d'ontologie OWL, est motivé par le fait que le langage OWL utilise la syntaxe XML et la sémantique est préservée par la majorité des éléments de XML.

Une interface entre le niveau sémantique et le niveau syntaxique est réalisée pour assurer le mécanisme de traduction des modules ontologique vers leurs homologues de base de données XML native.

Nous détaillerons les règles de transformation de notre modèle sémantique dans la section 5 qui utilise un exemple d'ontologie de la section 4.

4. Exemple d'ontologie

L'ontologie "courrier" illustrée par les figures (Fig 3.4, Fig 3.5, Fig 3.6) représente un modèle des services de courrier électronique, permettant à un émetteur qui est un service d'envoyer un message à un ou plusieurs destinataires (services). Les éléments de l'ontologie "courrier" sont exprimés par des classes (courrier, service, emetteur, destinataire), des propriétés de type de données porteuses des valeurs (objet, contenu, date_courrier, date_envoi, numero_courrier,…), et par les relations entre les différents éléments (Acommentaire, Aobjet…).

L'ontologie "courrier" est utilisée pour illustrer le processus de transformation entre une ontologie OWL et XML Schema. Elle est constituée en trois parties :

a- Les classes de l'ontologie courrier :

L'ontologie courrier comporte quatre classes (Fig 3.4) :

La classe "courrier": elle représente le courrier électronique assurant le service de transfert de messages envoyés d'un émetteur vers des destinataires. Un courrier est caractérisé par les propriétés : commentaire, objet, contenu,numéro courrier, date courrier.

La classe "emetteur" et classe "destinataire" sont des sous classe de la classe "service".

La classe "service" contient deux attributs nom service et code service.

La classe "type_destinataires" est une classe de type énumérée, elle contient les éléments "A" et "Cc".

Définition des classes de l'ontologie	
<owl><!--définition des classes--> <**owl:Class rdf:ID="courrier"**> <rdfs:subClassOf ><owl:Restriction> <owl:OnProperty rdf:resource="#commentaire"/> <owl:maxcardinality rdf:datatype="&xsd:int">1 <rdfs:maxcardinality /> <owl:Restriction/><rdfs:subClassOf /> <rdfs:subClassOf ><owl:Restriction> <owl:OnProperty rdf:resource="#objet"/> <owl:cardinality rdf:datatype="&xsd:int">1 <rdfs:cardinality />	<owl:OnProperty rdf:resource="#EnvoyéPar"/> <owl:cardinality rdf:datatype="&xsd:int">1 <rdfs:cardinality /> <owl:Restriction/><rdfs:subClassOf /> <rdfs:subClassOf ><owl:Restriction> <owl:OnProperty rdf:resource="#EnvoyéA"/> <owl:mincardinality rdf:datatype="&xsd:int">1 <rdfs:mincardinality /> <owl:Restriction/><rdfs:subClassOf /> <owl:Class/> <**owl:Class rdf:ID="emetteur"**>

```
  <owl:Restriction/><rdfs:subClassOf />
  <rdfs:subClassOf ><owl:Restriction>
    <owl:OnProperty rdf:resource="#contenu"/>
    <owl:cardinality rdf:datatype="&xsd:int">1
    <rdfs:cardinality />
  <owl:Restriction/><rdfs:subClassOf />
  <rdfs:subClassOf ><owl:Restriction>
    <owl:OnProperty rdf:resource=
      "# numéro_courrier "/>
    <owl:cardinality rdf:datatype="&xsd:int">1
    <rdfs:cardinality />
  <owl:Restriction/><rdfs:subClassOf />
  <rdfs:subClassOf ><owl:Restriction>
    <owl:OnProperty rdf:resource="#date_courrier"/>
    <owl:cardinality rdf:datatype="&xsd:int">1
    <rdfs:cardinality />
  <owl:Restriction/><rdfs:subClassOf />
  <rdfs:subClassOf ><owl:Restriction>
    <owl:OnProperty rdf:resource="#date_envoi"/>
    <owl:cardinality rdf:datatype="&xsd:int">1
    <rdfs:cardinality />
  <owl:Restriction/><rdfs:subClassOf />
  <rdfs:subClassOf ><owl:Restriction>
```

```
    <rdfs:subClassOf   rdf:resource="#service"/>
<owl:Class/>
<owl:Class rdf:ID="type_destinataires">
  <owl:oneOf rdf:parseType="Collection">
    <owl:Thing rdf:about="#A"/>
    <owl:Thing rdf:about="#Cc"/>
 </owl:oneOf></owl:Class>
<owl:Class rdf:ID="destinataire">
  <rdfs:subClassOf   rdf:resource="#service"/>
  <rdfs:subClassOf
rdf:resource="#type_destinataires"/>
<owl:Class/>
<owl:Class rdf:ID="service">
  <rdfs:subClassOf ><owl:Restriction>
    <owl:OnProperty  rdf:resource="#nom_service"/>
    <owl:cardinality rdf:datatype="&xsd:int">1
    <rdfs:cardinality />
  <owl:Restriction/><rdfs:subClassOf />
  <rdfs:subClassOf ><owl:Restriction>
    <owl:OnProperty  rdf:resource="#code_service"/>
    <owl:cardinality rdf:datatype="&xsd:int">1
    <rdfs:cardinality />
  <owl:Restriction/><rdfs:subClassOf />
<owl:Class/>
```

Fig 3.4. Classes de l'ontologie courrier

b- Les propriétés de l'ontologie courrier :

L'écriture des propriétés est l'étape qui va permettre de détailler les relations qui existent entre les classes (Fig 3.5).

Définition des propriétés d'objet de l'ontologie

```
<!--définition des propriétés d'objet-->
<owl:ObjectProperty rdf:ID="EnvoyéA">
  <rdfs:domain   rdf:resource="#courrier"/>
  <rdfs:range
rdf:resource="#destinataire"/>
<owl:DatatypeProperty/>
<owl:ObjectProperty rdf:ID="EnvoyéPar">
  <rdfs:domain   rdf:resource="#courrier"/>
  <rdfs:range      rdf:resource="#emetteur"/>
<owl:DatatypeProperty/>
```

```
<owl:ObjectProperty
rdf:ID="Atype_destinataire">
  <rdfs:domain
rdf:resource="#destinataire"/>
  <rdfs:range
rdf:resource="#type_destinataire"/>
<owl:DatatypeProperty/>
```

Fig 3.5. Propriétés d'objet de l'ontologie courrier

c- Les propriétés de type de données de l'ontologie courrier :

Ce sont ces propriétés qui permettent d'affecter des propriétés quantifiées aux instances des classes par exemple : objet, date_courrier, contenu, etc.

Définition des propriétés de type de données l'ontologie

```
<!--définition des propriétés de type de données-->
<owl:DatatypeProperty rdf:ID="commentaire">
  <rdfs:domain rdf:resource="#courrier"/>
  <rdfs:range rdf:resource="&xsd:string"/>
<owl:DatatypeProperty/>
<owl:DatatypeProperty rdf:ID="objet">
  <rdfs:domain rdf:resource="#courrier"/>
  <rdfs:range rdf:resource="&xsd:string"/>
<owl:DatatypeProperty/>
<owl:DatatypeProperty rdf:ID="coutenu">
  <rdfs:domain rdf:resource="#courrier"/>
  <rdfs:range rdf:resource="&xsd:string"/>
<owl:DatatypeProperty/>
<owl:DatatypeProperty rdf:ID="numero_courrier">
  <rdfs:domain rdf:resource="#courrier"/>
  <rdfs:range rdf:resource="&xsd:positiveinteger"/>
<owl:DatatypeProperty/>
<owl:DatatypeProperty rdf:ID="date_courrier">
  <rdfs:domain rdf:resource="#courrier"/>
<rdfs:range rdf:resource="&xsd:date"/>
<owl:DatatypeProperty/>
<owl:DatatypeProperty rdf:ID="date_envoi">
  <rdfs:domain rdf:resource="#courrier"/>
  <rdfs:range rdf:resource="&xsd:datetime"/>
<owl:DatatypeProperty/>
<owl:DatatypeProperty rdf:ID="code_service">
  <rdfs:domain rdf:resource="#service"/>
  <rdfs:range rdf:resource="&xsd:string"/>
<owl:DatatypeProperty/>
<owl:DatatypeProperty rdf:ID="nom_service">
  <rdfs:domain rdf:resource="#service"/>
  <rdfs:range rdf:resource="&xsd:string"/>
<owl:DatatypeProperty/>
<owl/>
```

Fig 3.6. Propriétés de type de données de l'ontologie courrier

5. Transformation de l'ontologie OWL vers XML Schema

Le langage d'ontologie OWL permet de formaliser un domaine des connaissances par la description des classes de ressources ou objets et leurs relations par la définition des propriétés [1]. L'objectif est de permettre de transformer automatiquement, un domaine de connaissance existant (partagé) ou conçu spécifiquement aux exigences d'une application, vers une base de données XML native modélisée sous le modèle XML Schema. Nous proposons ici un mécanisme de transformation des différents éléments du langage OWL vers les éléments appropriés du XML Schema [1][11][12], c'est-à-dire d'une structure sémantique

avec relations entre les éléments (classes et objets), vers une structure syntaxique sous forme d'une arbre XML Schema qui assure la persistance des données (Fig 3.7).

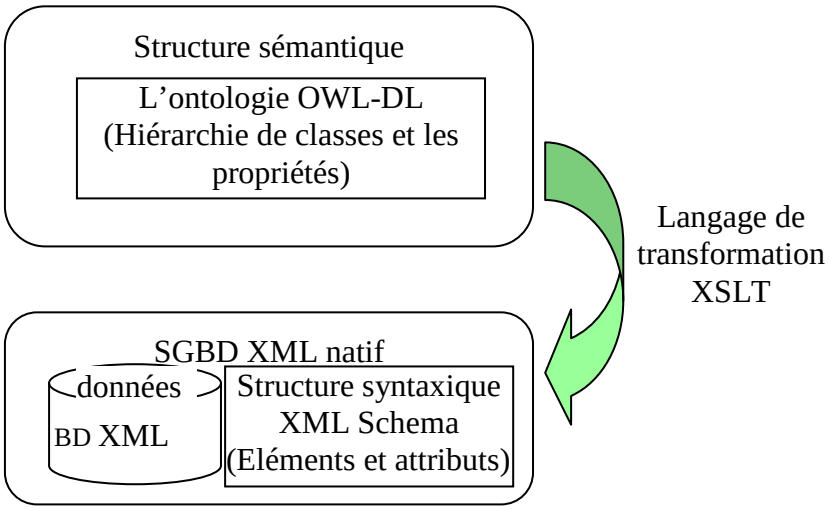


Fig 3.7. Principe de transformation de l'ontologie OWL-DL vers XML schema

Le modèle de données de XML décrit un arbre avec des nœuds étiquetés, quant à le modèle de données d'OWL est basé sur le triplet sujet-prédicat-objet de RDF et RDFSchema [1][4]. RDFS définit le vocabulaire d'un domaine et les relations entre les ressources ou objets de ce vocabulaire.

Nous voulons faire la correspondance entre les éléments du langage OWL (classes, propriétés d'objet, propriétés de type de données, restrictions de cardinalité…) et les éléments de la technologie XML Schema (types complexes, types simples, élément, attribut, cardinalités d'occurrences…) [1][12][11].

Le processus de transformation que nous proposons dans ce travail est inspiré de celui de Hannes Bohring et Soren Auer [11] qui traite la transformation automatique des éléments du modèle de données XML vers le langage d'ontologie OWL par la plateforme XSLT [13] afin de permettre de générer des ontologies de domaines à partir des documents XML. A l'inverse, notre mécanisme proposé, procède à la transformation automatique des éléments du langage d'ontologie OWL vers les éléments appropriés du XML Schema afin de permettre de stocker et manipuler les instances (données) de l'ontologie dans un contexte d'un SGBD XML natif. L'objectif de notre transformation est l'introduction de la sémantique dans les bases de données XML natives.

5.1. Transformation des classes d'OWL

Règle 1 : Les classes du langage OWL (owl :class) seront transformées vers le type complexe (xsd :complexType), à condition que la classe soit, au moins, un domaine de propriétés (rdfs :domain) du propriété de type de donnée (owl :DatatypeProperty).

Exemple :

La classe courrier en OWL:

```
<owl:Class rdf:ID="courrier">
  …
<owl:Class/>
```

Le type complexe courrier équivalent à la classe courrier d'OWL:

```
<xsd:complexeType name="courrier">
    <xsd:sequence>
      …les éléments et les attributs
    <xsd:sequence/>
<xsd:complexeType/>
```

Les autres classes d'OWL relatives à notre exemple qui vont être traduites vers un type complexe sont: la classe "service" et la classe "destinataire".

Règle 2 : Nous remarquons que la classe "emetteur" n'est pas transformée vers un type complexe de XML Schema parce que elle n'est un domaine de propriétés (rdfs :domain) d'une propriété de type de donnée (owl :DatatypeProperty). Elle est juste une sous classe de la classe "service". Donc elle va être un élément de type complexe du type complexe "service".

5.2. Transformation des classes d'OWL définies par énumération des instances

Règle 3 : Les classes définies par énumération de ses instances seront transformées vers le type simple (xsd:simpleType).

Exemple :

La classe "type_destinataire" d'OWL:

```
<owl:Class rdf:ID="type_destinataire">
        <owl :oneOf  rdf :parseType := "Collection">
            <owl:Thing  rdf:about="A">
            <owl:Thing  rdf:about="Cc">
    </owl :oneOf>
</owl:Class>
```

Le type simple "type_destinataire" équivalent à la classe "type_destinataire" d'OWL:

```
<simpleType name="type_destinataire">
  <restriction base="string">
   <enumeration value="A"/>
   <enumeration value="Cc"/>
  </restriction>
 </simpleType>
```

5.3. Transformation des propriétés type de donnée d'OWL

Les classes du langage OWL (owl :class) déjà transformées vers le type complexe (xsd :complexType) devront contenir au moins un élément de type simple ou même de type complexe ou un attribut, et surtout associer les types (integer, string, …) aux éléments. Donc comment localiser les éléments et attributs des types complexes.

Règle 4 : Toute propriété de type de donnée de l'ontologie (owl :DatatypeProperty) où le domaine de définition de propriété (rdfs :domain) est la classe transformée à un type complexe (xsd :complexType), sera une déclaration d'élément simple (xsd :element) de ce type complexe.

Règle 5 : La propriété type de donnée relie, les éléments d'une classe à un ensemble de valeurs de type prédéfini (rdfs:range). Le langage OWL utilise la plupart des types de données intégrés du schéma XML. Ceci facilite la conservation des types de données des différents éléments transférés.

Exemple :

La propriété type de donnée "commentaire" :

```
<owl:DatatypeProperty rdf:ID="commentaire">
  <rdfs:domain   rdf:resource="#courrier"/>
  <rdfs:range    rdf:resource="&xsd:string"/>
<owl:DatatypeProperty/>
```

La propriété type de donnée "commentaire" de l'ontologie OWL a comme domaine de définition la classe "courrier". Alors cette propriété va être définie comme un élément du type complexe "courrier" de XML Schema. Aussi comme il est défini dans l'ensemble de valeurs (rdfs:range) que la propriété type de donnée "commentaire" va prendre des valeurs de types chaîne de caractère (string) dans la définition de l'élément.

```
<xsd:complexType name="courrier">
<xsd:sequence>
        <xsd:element name="commentaire" type="xsd:string"/>
        ……
</xsd:sequence>
</xsd: complexType>
```

Les autres propriétés type de donnée seront transformées chacune dans son type complexe approprié :

```
<xsd:complexType name="courrier">
<xsd:sequence>
        <xsd:element name="commentaire" type="xsd:string"/>
        <xsd:element name="objet" type="xsd:string"/>
        <xsd:element name="contenu" type="xsd:string"/>
        <xsd:element name="numero_courrier" type="xsd:positiveinteger"/>
        <xsd:element name="date_courrier" type="xsd:date"/>
        <xsd:element name="date_envoi" type="xsd:datetime"/>
</xsd:sequence>
```

```
</xsd: complexType>
<xsd:complexType name="service">
<xsd:sequence>
        <xsd:element name="code_service" type="xsd:string"/>
        <xsd:element name="nom_service" type="xsd:string"/>
</xsd:sequence>
</xsd: complexType>

<xsd:complexType name="destinataire">
<xsd:sequence>
        <xsd:element name="type_destinataire" type="xsd:string"/>
</xsd:sequence>
</xsd: complexType>
```

5.4. Transformation des propriétés d'objet d'OWL

Règle 6 : Toute propriété d'objet de l'ontologie (owl :ObjectProperty) où le domaine de définition de propriété (rdfs :domain) est la classe transformée à un type complexe (xsd :complexType) sera traitée par le processus de transformation comme suit: La ressource de l'image (rdfs :range) sera une déclaration d'élément (xsd :element) de type complexe de ce type complexe transformée à partir de la classe de domaine de définition de la propriété d'objet.

Exemple :

a) La propriété d'objet "EnvoyéA" :

```
<owl:ObjectProperty rdf:ID="EnvoyéA">
  <rdfs:domain   rdf:resource="#courrier"/>
  <rdfs:range      rdf:resource="#destinataire"/>
<owl:DatatypeProperty/>
```

La propriété d'objet "EnvoyéA" a pour domaine la classe "courrier" et pour image la ressource "destinataire", c'est-à-dire qu'elle relie des instances de la classe "courrier" à des instances de la ressource "destinataire". Alors la ressource "destinataire" sera une déclaration d'élément (xsd :element) de type complexe de ce type complexe "courrier".

```
<xsd:complexType name="courrier">
<xsd:sequence>
        …….
        <xsd:element name="destinataire" type="?????"/>
</xsd:sequence>
</xsd: complexType>
```

b) Règle 7 : S'il existe une classe transformée vers un type complexe, et ce type complexe a le même nom de l'élément d'un autre type complexe, nous renommons le nom du premier type complexe en rajoutant la lettre "s". Donc le type complexe "destinataire" développé auparavant lors de la transformation des classes sera:

```
<xsd:complexType name="destinataires">
<xsd:sequence>
        <xsd:element name="type_destinataire" type="xsd:string"/>
</xsd:sequence>
</xsd: complexType>
```

La ressource "destinataire" sera une déclaration d'élément (xsd :element) de type complexe "courrier" aura le type complexe "destinataires".

```
<xsd:complexType name="courrier">
<xsd:sequence>
```

…….

<xsd:element name="destinataire" ref="destinataires"/>

</xsd:sequence>

</xsd: complexType>

La propriété d'objet "EnvoyéPar" :

<owl:ObjectProperty rdf:ID="EnvoyéPar">

<rdfs:domain rdf:resource="#courrier"/>

<rdfs:range rdf:resource="#emetteur"/>

<owl:DatatypeProperty/>

La propriété d'objet "EnvoyéPar" a pour domaine la classe "courrier" et pour image la ressource "emetteur". C'est-à-dire qu'elle relie des instances de la classe "courrier" à des instances de la ressource "emetteur". Alors la ressource "emetteur" sera une déclaration d'élément (xsd :element) de type complexe de ce type complexe "courrier".

c) Règle 8 : S'il existe une classe transformée vers un type complexe, et l'élément d'un autre type complexe est une sous classe de la première classe, nous référençons l'élément au type complexe de la sous classe transformée.

<xsd:complexType name="courrier">

<xsd:sequence>

…….

<xsd:element name="destinataire" ref="destinataires"/>

<xsd:element name="emetteur" ref="service"/>

</xsd:sequence>

</xsd: complexType>

5.5. Transformation des cardinalités d'OWL

Règle 9 : XML Schema comporte aussi des contraintes d'arité comme xsd:minOccurs et xsd:maxOccurs, qui permettent de déterminer le nombre d'occurrences d'un élément ou attributs dans un document XML. Ces contraintes sont équivalentes à celle du langage d'ontologie OWL (owl:mincardinality, owl:maxcardinality et owl:cardinality) qui permettent d'exprimer le nombre d'éléments dans une relation.

Exemple :

La contrainte de cardinalité "maxcardinality" :

```
<owl:Class rdf:ID="courrier">
<rdfs:subClassOf ><owl:Restriction>
        <owl:OnProperty rdf:resource="#commentaire"/>
            <owl:maxcardinality rdf:datatype="&xsd:int">1
            <rdfs:maxcardinality />
    <owl:Restriction/><rdfs:subClassOf />
    <rdfs:subClassOf ><owl:Restriction>  …
 <owl:Class/>
```

Dans cet exemple la contrainte de cardinalité "maxcardinality" toute seule signifie qu'un courrier possède au maximum un commentaire. Si la contrainte "mincardinality" n'est pas spécifiée, alors il est possible alors qu'un courrier n'a aucun commentaire.

Le commentaire est optionnel dans le type complexe courrier par le biais de la valeur "0" de l'attribut minOccurs. En général, un élément est requis quand la valeur de minOccurs vaut "1" ou plus:

```
<xsd:complexType name="courrier">
<xsd:sequence>
        <xsd:element name="commentaire" type="xsd:string" minOccurs="0"
          maxOccurs="1"/>
        ……..
</xsd:sequence>
```

</xsd: complexType>

Le nombre maximum de fois qu'un élément peut apparaître est déterminé dans sa déclaration par la valeur de l'attribut maxOccurs , ou encore le mot "unbounded" qui signifie qu'il n'y a pas de valeur limite. La valeur par défaut dans les deux cas (minOccurs et maxOccurs) est "1".

Si la contrainte "cardinality" vaut "1" dans l'ontologie OWL, alors nous utilisons la valeur par défaut (minOccurs et maxOccurs) qui est "1".

<xsd:complexType name="courrier">

<xsd:sequence>

…… <xsd:element name="contenu" type="xsd:string"/> ……..

</xsd:sequence>

</xsd: complexType>

Le contenu est obligatoire et une seule fois dans le type complexe courrier par le biais de la valeur par défaut des attributs minOccurs et maxOccurs.

5.6. Obtention du schéma XML après transformation

La correspondance entre les éléments du langage OWL (classes, propriétés d'objet, propriétés de type de données, restrictions de cardinalité…) et les éléments de la technologie XML Schema (types complexes, types simples, élément, attribut, cardinalités d'occurrences…) nous a permis d'obtenir le fichier courrier.xsd comme il est illustrée dans la figure (Fig 3.8).

Le schéma de courrier courrier.XSD

```
<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema">
 <xsd:annotation>
  <xsd:documentation xml:lang="fr">
   schéma du document courrier créé par BOURBIA Tarek 09/11/07…
  </xsd:documentation>
 </xsd:annotation>

 <xsd:complexType name="courrier">
  <xsd:sequence>
   <xsd:element name="emetteur" ref="service"/>
   <xsd:element name="destinataire" ref="destinatires" minOccurs="0"
           maxOccurs="unbounded"/>
   <xsd:element name="objet" type="xsd:string"/>
   <xsd:element name="contenu" type="xsd:string"/>
   <xsd:element name="numero_envoi" type="xsd:myInteger"/>
   <xsd:element name="date_envoi" type="xsd:date"/>

   <xsd:element name="commentaire" type="xsd:string" minOccurs="0"
           maxOccurs="1"/>
    </xsd:complexType>

 <xsd:complexType name="service">
  <xsd:sequence>
   <xsd:element name="code_service"   type="xsd:string"/>
   <xsd:element name="nom_service" type="xsd:string"/>
  </xsd:sequence>
</xsd:complexType>

 <xsd:complexType name="destinataires">
  <xsd:sequence>
     <simpleType name="type_destinataire">
          <restriction base="string">
              <enumeration value="A"/>
              <enumeration value="Cc"/></restriction></simpleType>
    <xsd:element ref=="service" />
  </xsd:complexType>
</xsd:schema>
```

Fig 3.8. Fichier XML Schema obtenu

6. Transformation des requêtes sémantiques vers des requêtes syntaxiques

Le modèle relationnel et le modèle d'arbre XML sont largement utilisés pour représenter les données structurées et semi structurées, mais la sémantique est absente dans ces deux

modèles. OWL avec ses concepts offre une sémantique et un sens aux données et permet, aussi, l'utilisation de langages de requêtes sémantiques.

Les instances (données) des classes OWL sont chargées aux nœuds de l'arbre XML, tout en respectant le schéma XML obtenu par le processus de transformation du modèle logique en ontologie OWL vers la base de données XML native sous la structure XML Schema. Les instances des classes OWL sont les individus des attributs d'un document XML ou les individus des éléments d'un document XML qui n'ont pas des fils éléments ou attributs.

SPARQL fournit un moyen puissant pour l'extraction des informations à partir d'un ensemble des instances d'une ontologie rangé sous des triplets RDF. Et pour le faire avec une ontologie OWL-DL construise au dessus d'une base de données XML native, il faut utiliser un processus de transformation des différentes clauses, de la requête de recherche sémantique SPARQL, vers les clauses appropriées de la requête XQUERY du SGBD XML natif.

Notre approche utilisée pour assurer cette transformation consiste à s'en servir d'un intermédiaire qui est l'algèbre relationnelle[10] pour faire la correspondance entre les clauses de deux langages de requêtes SPARQL et XQUERY (Fig 3.9).

De ce fait, nous proposons un processus de traduction d'une requête sémantique formulée avec SPARQL en une requête construite avec les opérateurs algébriques. Cette dernière sera par la suite traduite en une requête supportée par le SGBD XML natif basée sur XQUERY.

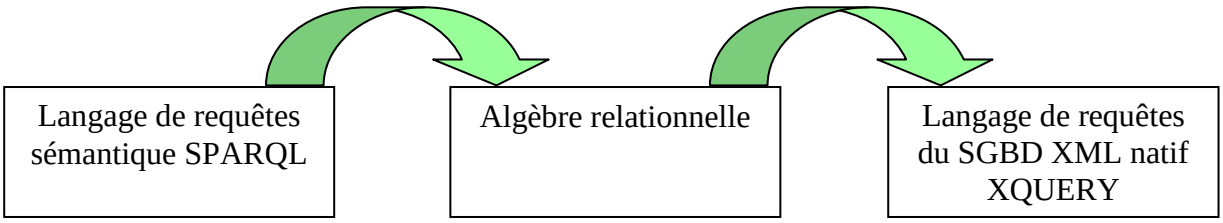


Fig 3.9. Transformation des requêtes par l'utilisation d'un intermédiaire

L'avantage de transformation des requêtes SPARQL en algèbre relationnelle est :

- De permettre l'optimisation de la requête SPARQL, avant même l'optimisation offerte, au niveau des requêtes XQUERY, par le SGBD XML natif.

[10] L'algébre relationnelle est un langage déclaratif constitue d'opérateurs unaires et binaires qui permettent les manipulations des tables relationnelles.

- De permettre de supporter les requêtes SPARQL par n'importe quel langage de requête s'appuyant sur l'algèbre relationnelle.

La requête suivante exprimée en langage SPARQL interroge la base de données à base ontologique construite sur l'ontologie OWL-DL (Fig 3.4) (Fig 3.5) (Fig 3.6). Elle donne comme résultat l'objet et le nom du service émetteur des courriers envoyés après la date du 01/01/2009. L'arbre algébrique de cette requête est illustré dans la figure (Fig 3.10)

```
SELECT ?objet, ?nom_service as service_emetteur
WHERE {?courrier rdf:type courrier
        ?courrier objet ?objet
          ?courrier date_envoi ?date_envoi
          FILTER ( ?date_envoi>=01/01/2009)
        ?courrier envoyéPar ?emetteur
        ?emetteur rdfs:subclass service
        ?emetteur nom_service ?nom_service}
```

L'équivalent de cette requête SPARQL en langage de requête du SGBD XML natif XQUERY est :

```
FOR $a in document(courrier.xml)
WHERE $a/courrier/date_envoi>=01/01/2009
RETURN $a/courrier/objet, $a/emetteur/nom_service
```

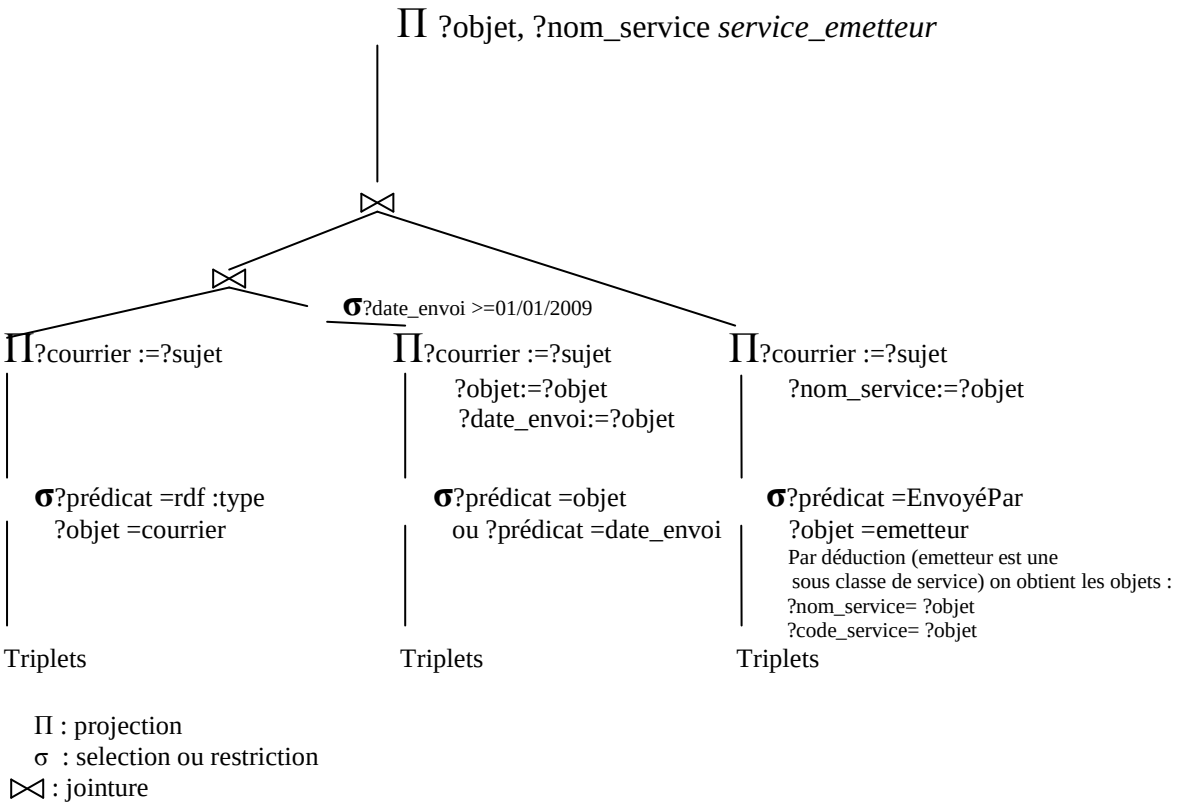


Fig 3.10. Arbre de l'algèbre relationnelle pour une requête SPARQL

6.1. Algèbre relationnelle pour les requêtes SPARQL

L'algèbre relationnelle est un langage formel comporte des opérateurs unaires et binaires, permettant la recherche et l'extraction des données des tables du modèle relationnel. Son principe repose sur la création de nouvelles tables résultantes à partir de la combinaison de six opérateurs de base : sélection, projection, jointure, renommer, union, différence [41].

SPARQL est un langage de requêtes pour l'interrogation de métadonnées et l'extraction des données sous forme de triplets RDF de l'ontologie. Un triplet RDF est une association: **{sujet, prédicat, objet}**.

La syntaxe du SPARQL est inspirée du SQL. Il permet d'exprimer l'extraction par SELECT des sous graphes, à partir du graphe RDF, satisfaisant les conditions spécifiées dans la clause WHERE. Ce qui nous a permis de proposer dans cette section une adaptation des opérateurs de base de l'algèbre relationnelle avec les différentes clauses et composants de clauses du langage de requête de recherche sémantique SPARQL.

6.1.1. Sélection

La sélection (**σ**) ou la restriction est une opération unaire sélectionne des tuples au moyen d'un prédicat, et produit un résultat de même schéma mais ne comportant que les tuples satisfaisant la condition précisée en argument. Le prédicat ou la condition de sélection est évaluée à vrai ou faux pour chaque tuple.

Sa syntaxe : $\sigma_{<\text{prédicat-sélection}>}$ (expression algébrique)

Où expression algébrique est une expression qui résulte de l'évaluation de l'expression.

Dans le langage de requête SPARQL le prédicat de sélection est formulé par deux manières :

1. La clause FILTER qui applique la condition sur les instances (données), par exemple : FILTER (?date_envoi>=01/01/2009). En algèbre relationnelle du langage SPARQL, cette clause est formulée par : $\sigma_{?\text{date\_envoi} >= 01/01/2008}$ (T), dont T est les triplets de l'ontologie. Ici la formule est évaluée par l'instanciation de la variable ?date_courrier avec chaucun des tuples existants. C'est exactement comme la formulation d'une sélection algébrique en SQL par la clause WHERE.

2. La restriction qui s'applique sur les propriétés et les objets de l'ontologie en spécifiant un prédicat. Dans ce type de restriction le prédicat de sélection peut être d'une part, une propriété de type de donnée de l'ontologie, par exemple : ?courrier date_envoi ?date_envoi (Fig 3.10). Et d'autre part, un type de propriété, par exemple : ?courrier EnvoyéPar ?emetteur (Fig 3.10).

La deuxiéme maniére de formuler le prédicat de sélection, que ce soit pour le type de propriété(Proprety) ou pour la propriété type de données(DatatypeProperty), est exprimée en algèbre relationnelle du langage de requête SPARQL par : $\sigma_{\text{prédicat = date\_envoi}}$. Nous considérons ici le mot prédicat comme un mot clé.

Le sujet et l'objet du prédicat de type de propriété sont des classes, de ce fait, nous pouvons distinguer deux cas :

1. Les classes disposant des objets explicites de type propriété de type de données : Tous les objets de ces classes seront des sujets ou des objets satisfaisant cette sélection par prédicat, par exemple la sélection $\sigma_{\text{prédicat =date\_envoi}}$ (Fig3.10) donne comme résultat l'objet date_envoi de la classe courrier. Les classes disposant d'un objet propriété type de données, qui repésente les instances (données) d'une

manière unique sera l'objet ou le sujet de cette classe par défaut, par exemple le sujet de la classe courrier est implicitement l'objet "numéro_courrier" (Fig 3.10).

2. Les classes qui ne disposant pas des objets explicites de type propriété de type de données : Les objets de ce type de classes seront déduits à partir d'un mécanisme d'inférence par héritage et transitivité à partir des super classes, par exemple les objets "code_service" et "nom_service" de la classe "emetteur" sont déduits à partir de sa super classe "service"(Fig 3.10).

6.1.2. Projection

La projection (Π) est une opération unaire, produit un résultat de schéma différent en ne conservant qu'un sous ensemble des attributs (objets et sujets des triplets de l'ontologie) mentionnés en opérandes et les tuples correspondants en éliminant les doublons.

Sa syntaxe : Π<liste attributs> (expression algébrique)

Où l'expression algébrique est une expression qui résulte de l'évaluation de l'expression.

La projection dans le langage de requêtes SPARQL, ne concerne que les sujets et les objets qui sont des propriétés de type de données (DataTypeProperty) de l'ontologie OWL. Le sujet des triplets dans ce contexte est une classe qui est représentée par une propriété de type de données qui prend des valeurs uniques, comme la clé primaire dans le modèle relationnel. Par exemple pour la classe "courrier" nous choisissons implicitement l'objet "numéro_courrier" comme sujet des triplets. Ce choix est primordial parce qu'il servira pour la combinaison entre les tuples lors de l'opération de jointure.

Dans l'arbre de l'algèbre relationnelle de la requête SPARQL, illustrée dans la figure (Fig 3.10), il existe quatre projections :

1. Π ?courrier :=?sujet : c'est une projection sur le sujet ?courrier qui est type de la classe "courrier". Le sujet ?courrier projecté est implicitement l'objet "numéro_courrier" qui représente les instances (données) d'une manière unique.

2. Π ?courrier :=?sujet et ?objet:=?objet et ?date_envoi:=?objet : ici nous avons fait la projection sur le sujet (?courrier) comme celle ci-dessus ; et sur deux objets (?objet : objet du courrier, ?date_envoi : date émission du courrier).

3. Π ?courrier :=?sujet et ?nom_service:=?objet : comme les projections précédente nous avons projeté le sujet (?courrier), l'autre partie de la projection concerne l'objet ?nom_service qui est l'emetteur de courrier.

4. Π ?objet :=?objet et ?nom_service:=?objet : C'est le résultat de notre requête SPARQL. C'est la clause SELECT de la requête qui renvoie comme résultat ici l'objet du courrier et son service émetteur.

6.1.3. Renommage

Le renommage est une opération unaire, permet la création d'alias pour des objets de type propriété de type de données de l'ontologie. Lorsque le nom de l'objet est remplacé par un alias, ce dernier devient le nom temporaire de cet objet dans la réponse. Le renommage est utilisé ici pour donner plus de sens lors de l'affichage du résultat.

Par exemple dans la figure (Fig 3.10), l'objet ?nom_service est renommé avec l'alias service_emetteur qui le remplace dans l'affichage du résultat. L'expression algébrique ?nom_service *service_emetteur* assure ce renommage. L'alias service_emetteur a plus de sens que ?nom_service. ?nom_service peut être interprété comme service émetteur ou service destinataire du courrier.

6.1.4. Jointure

La jointure (|**x**|) est une opération binaire, produit à partir de deux ensembles d'objets et de sujets, un ensemble de schéma différent dont les tuples sont obtenus en composant un tuple de premier ensemble et un tuple de deuxiéme ensemble lorsque ceux-ci ont la même valeur d'objet en commun ou de même nom. S'il n'y a pas des objets en commun, l'effet de la jointure est un produit cartésien.

Sa syntaxe : ressources1 $|x|_p$ ressources2

ressources sont les sujets et les objets obtenus à partir des triplets de l'ontologie ou par l'opération algébrique de projection.

p c'est le filtre qui précise l'objet ou le sujet de même nom sur lequel la jointure s'applique.

Dans l'arbre de l'algèbre relationnelle de la requête SPARQL, illustrée dans la figure (Fig 3.10), il existe deux jointures :

1. J1=Π?courrier(**σ**prédicat=rdf :type et ?objet=courrier(triplets)) |**x**| **σ** ?date_envoi>=01/01/2009 (Π ?courrier, ?objet, ?date_envoi (**σ**prédicat =objet ou prédicat=date_envoi(triplets))) : c'est une jointure sur le sujet en commun ?courrier qui est type de la classe "courrier" et qui donne comme résultat le numéro, l'objet et la date d'envoi des courriers envoyés après la date du 01/01/2009. Le sujet ?courrier projeté est implicitement l'objet

"numéro_courrier". La partie gauche de cette jointure contient un seul objet projeté qui est le sujet "numéro_courrier" obtenu par l'opération de sélection des objets de la classe "courrier" ; la partie droite contient le sujet en commun "numéro_courrier" et les deux objets "objet" et "date_envoi" de courriers envoyés après 01 janvier 2009.

2. J2=J1|**x**| Π?courrier, ?nom_service(**σ** ?prédicat =EnvoyéPar et ?objet =emetteur(triplets)) : c'est cette jointure sur le sujet en commun ?courrier qui renvoie la réponse de l'exécution finale de la requête SPARQL, elle donne comme résultat l'objet et le nom de service émetteur des courriers envoyés après la date du 01/01/2009. Le sujet ?courrier projeté est implicitement l'objet "numéro_courrier". La partie gauche de cette jointure est le résultat de la jointure J1 ci-dessus ; la partie droite contient le sujet en commun "numéro_courrier" et l'objet "nom_service" émetteur de courrier.

6.1.5. Union

L'union (U) est une opération binaire, produit à partir de deux ensembles d'objets et de sujets de même schéma, un ensemble résultant de la même composante ayant pour tuples ceux appartenant au deux ou à un ensemble en éliminant les doublons éventuels.

6.1.6. Différence

La différence (-) est une opération binaire, produit à partir de deux ensembles d'objets et de sujets de même schéma, un ensemble résultant de la même composante ayant pour tuples ceux appartenant au premier ensemble mais pas pour le deuxième.

6.1.7. Règles de transformation de SPARQL vers l'algèbre relationnelle

La requête SPARQL est basée sur les motifs de triplet appelés motifs de graphe élémentaire. Chaque triplet est une association: **{sujet, prédicat, objet}**, par exemple :

sujet=?courrier **prédicat**=date_envoi **objet**=?date_envoi.

La syntaxe des requêtes SPARQL basées sur le motif de graphe est composée généralement de :

1. La clause SELECT : les variables qui doivent apparaître dans les résultats.
2. La clause WHERE : les motifs de graphe élémentaire.
3. La clause FILTER : restriction de résultat par des expressions régulières.

Pour transformer une requête SPARQL en opérations de l'algèbre relationnelle, nous proposons les règles suivantes :

Règle 1 : Chaque triplet {sujet, prédicat, objet} du SPARQL est transféré en algèbre relationnelle par l'utilisation d'une sélection par le prédicat suivie d'une projection sur le sujet et l'objet, par exemple :

?courrier date_envoi ?date_envoi. En algébre : $\Pi_{?courrier,?date\_envoi}(\sigma_{prédicat =date\_envoi(triplets)})$

Ou $\Pi_{?numéro\_courrier,?date\_envoi}(\sigma_{prédicat =date\_envoi(triplets)})$, parce que le sujet ?courrier est type de la classe "courrier". Le sujet ?courrier projeté est implicitement l'objet "numéro_courrier" qui représente les instances (données) d'une manière unique de cette classe.

Cette règle n'est valable que pour les sujets et les objets qui sont de type propriété de type de donnée (DatatypePreperty) de l'ontologie OWL-DL et l'attribut du prédicat n'appartient pas au vocabulaire du langage OWL-DL.

Règle 2 : Si l'attribut du prédicat appartient au vocabulaire du langage OWL-DL, un processus d'inférence est déclenché pour faire sortir tous les sujets et les objets déductibles, qui par la suite vont être projetés, par exemple :

?destinataire rdfs :subclassof "service". En algèbre l'expression se formule comme $\Pi_{?code\_service,?nom\_service}$, où les objets ?code_service et ?nom_service sont des propriétés de type de donnée de la classe supérieure "service".

Règle 3 : Si l'objet ou le sujet du triplet est une classe qui ne contient aucun objet de type propriété de type de donnée, une inférence est déclenchée pour faire sortir tous les objets de de cette classe par héritage ou transitivité, par exemple : courrier envoyéPar ?emetteur. En algèbre $\Pi_{?code\_service,?nom\_service.}$ "emetteur" est une classe qui ne dispose pas des objets explicites de type propriété de type de données, les objets de cette classe seront déduits à partir de l'inférence par héritage et transitivité à partir de sa super classe, par exemple les objets "code_service" et "nom_service" de la classe sont déduits à partir de sa super classe "service".

Règle 4 : La clause FILTER du SPARQL qui applique la condition sur les données, sera transformée en opération algébrique de sélection, par exemple : FILTER (?date_envoi>=01/01/2009). En algèbre relationnelle du langage SPARQL cette clause est

formulée par : $\sigma_{?date\_envoi >=01/01/2008}$ (triplets). C'est exactement comme la formulation d'une sélection algébrique en SQL pour la clause WHERE.

Règle 5 : Les premières jointures se font au niveau des triplets {sujet, prédicat, objet} deux par deux sur un objet ou un sujet en commun ou de même nom. Elle permet d'obtenir un ensemble des objets et des sujets. Ensuite elle se complique en montant, parce qu'elle s'effectue à partir de deux ensembles d'objets et de sujets.

L'ordre d'organisation des jointures n'est pas important, parce que la jointure est commutative (Jointure1 |x| Jointure2) et associative (Jointure1 |x| Jointure2) |x| Jointure3 = Jointure1 |x| (Jointure2 |x| Jointure3).

Dans la requête SAPRQL suivante :

SELECT ?objet, ?nom_service as service_emetteur WHERE

{1 ?courrier rdf:type courrier

2 ?courrier objet ?objet

3 ?courrier date_envoi ?date_envoi

4 FILTER (?date_envoi>=01/01/2009)

5 ?courrier envoyéPar ?emetteur

6 ?emetteur rdfs:subclass service

7 ?emetteur nom_service ?nom_service}

Nous pouvons établir des jointures entre le tuple de la ligne1 avec celui de la ligne2 sur l'objet de ?courrier (?numéro_courrier). Le résultat de chaque jointure entre automatiquement avec la ligne suivante s'il y a un objet ou un sujet en commun et en ignorant les clauses FILTER.

Les clauses FILTER transformée en opération algébrique de sélection, peuvent être introduites après ou avant une jointure. La meilleure position à déterminer entre dans la phase de l'optimisation de la requête.

Règle 6 : La partie SELECT de la requête SPARQL est transformée en algèbre relationnelle par une projection des objets et sujets de l'expression algébrique obtenue de la clause WHERE, par exemple : SELECT ?objet, ?nom_service as service_emetteur. En algèbre relationnelle : $\Pi_{?objet,\ ?nom\_service\ as\ service\_emetteur}$(expression algébrique de la requête)

Le renommage est intégré ici avec la projection.

6.2. Règles de transformation de l'algèbre relationnelle vers XQUERY

XQUERY et SPARQL ressemblent au SQL. Le premier permet de réaliser des requêtes dans des documents XML et le deuxième dans des triplets RDF de l'ontologie. Nous pouvons trouver des similitudes, juste au niveau de la syntaxe.

XQuery parcourt des données hiérarchiques, pour obtenir des éléments particuliers dans un arbre, nous devons indiquer le document puis ajouter les éléments ou attributs souhaités.

La requête suivante exprimée en langage XQUERY interroge la base de données XML native et retourne comme résultat l'objet et le nom de service émetteur des courriers envoyés après la date du 01/01/2009 :

```
FOR $a in document(courrier.xml)
WHERE $a/courrier/date_envoi>=01/01/2009
RETURN $a/courrier/objet, $a/courrier/emetteur/nom_service
```

Nous allons faire une transformation de l'expression de l'algèbre relationnelle, obtenue de la requête de recherche sémantique SPARQL, vers la syntaxe du langage de requête XQUERY. De ce fait, nous proposons dans cette section une adaptation des opérateurs de base de l'algèbre relationnelle avec les différentes clauses du langage de requête XQUERY. Cette transformation est dépendante de schéma XML (XSD) afin de permettre de déterminer le chemin des éléments et attributs du document XML par le langage XPATH (langage de navigation dans l'arbre XML).

6.2.1. Chemin des éléments et des attributs

Avant de faire la correspondance entre les opérateurs de l'algèbre relationnelle et les composants des clauses de XQUERY, nous devons d'abord déterminer pour chaque objet de l'ontologie ou attribut des expressions algébriques son équivalent en chemin XPATH, en se basant sur le schéma XML. L'objectif de XPATH est de localiser des parties de documents XML à l'aide d'expression de chemin.

Les expressions de XPATH sont similaires de celles utilisées par les systèmes de fichiers, par exemple: /courrier/emetteur/nom_service. Ici XPATH navigue dans le document XML courrier.xml, et sélectionne les noms des services émetteurs des courriers.

La règle principale pour déterminer les chemins XPATH qui vont être utilisés dans les requêtes XQUERY est que chaque élément ou attribut d'un type complexe de la racine doit avoir la représentation : nom_type_complexe/nom_élément

Et si l'élément du type complexe est de type complexe ou se réfère à un type complexe la représentation devient nom_type_complexe/ nom_type_complexe1/nom_élément1

Nous continuons en profondeur jusqu'à l'arrivée aux éléments de type simple.

Par exemple le chemin XPATH de l'objet nom_service émetteur de courrier est : $a/courrier/emetteur/nom_service ou $a est le document XML de notre base de données XML.

6.2.2. Sélection

L'opérateur (σ) de sélection est transformé par deux manières selon la nature du prédicat :

1. Le prédicat qui prend des valeurs booléennes comme l'expression: $\sigma_{?date\_envoi >= 01/01/2008}$ de la figure (Fig 3.10), il sera considéré comme une condition de filtrage et sera une partie de la clause WHERE de la requête XQUERY, comme suit : WHERE $a/courrier/date_envoi>=01/01/2009.

2. Le prédicat qui est une restriction qui s'applique sur les propriétés et les objets de l'ontologie : Dans ce type de restriction les objets et les sujets des prédicats seront transformés en expressions de chemin XPATH appropriées, par exemple l'objet ?nom_service est transformé en chemin /courrier/emetteur/nom_service dans la requête XQUERY.

6.2.3. Projection

Nous considérons ici que la base de données XML native est constituée d'un seul composant, de ce fait, la seule projection que nous considérons de l'expression algébrique est celle la plus à gauche, et ces objets et sujets seront une partie de clause RETURN de la requête XQUERY, après bien sûre la transformation de chaque élément en chemin XPATH.

Par exemple : $\Pi_{?objet,\ ?nom\_service\ as\ service\_emetteur}$(expression algébrique de la requête) sera transformée en RETURN $a/courrier/objet, $a/courrier/emetteur/nom_service.

6.2.4. Renommage

Le renommage en XQUERY n'existe pas, de ce fait, nous ignorons tout renommage dans l'expression algébrique. Le renommange n'est pas nécessaire parce que tous les éléments sont représentés par un chemin explicite XPATH.

6.2.5. Jointure

Dans notre SGBD XML natif, nous considérons que la base de données est constituée d'un seul arbre XML, et pour cela la jointure XQUERY entre plusieurs arbres n'est pas illustrée dans notre travail. Néanmoins, si l'expression algébrique contient un opérateur de jointure c'est à dire une requête SPARQL à partir de plusieurs ontologies, elle sera transformée comme démontrée dans l'exemple suivant :

Expression algébrique : $\Pi_{?objet1,\ ?objet2}(T.O1) \mid \mathbf{x} \mid \Pi_{?objet1,\ ?objet3}(T.O2)$

Requête XQUERY : FOR $a in document(courrier.xml)

$b in document(document.xml)

WHERE $a/courrier/id_courrier=$b/document/id_document

C'est une jointure entre le document courier.xml et document.xml de la même base de données XML sur l'identifiant en commun.

6.2.6. Union et Différence

Les deux opérateurs binaires l'union et la différence sont simples à transférer en requête XQUERY. L'expression algébrique exp1 U exp1 sera en XQUERY : requête1 U requête2 où les expressions algébriques exp1 et exp2 seront transformées en XQUERY requête1 et requête2.

7. Conclusion

Dans ce chapitre nous avons proposé une architecture étendant les fonctionnalités d'un SGBD XML natif par une couche représentant un niveau sémantique exprimée par le langage d'ontologie OWL-DL, afin d'avoir une représentation plus expressive et de permettre l'inférence sur les bases de données. Nous avons aussi cité et défini les deux composants de cette couche :

1. le module des connaissances sous le langage d'ontologie OWL-DL.
2. le module de la prise en charge des requêtes (de recherche) sémantique sous le langage de requête sémantique SPARQL.

Nous pouvons qualifier le cadre de travail présenté dans ce chapitre par la notion de Bases de Données à Base Ontologique, qui se situe dans l'intersection de deux domaines : les bases de données et les ontologies.

Pratiquement, l'architecture proposée concerne deux axes différents:

1. un processus de transformation des éléments du langage OWL-DL (classes, propriétés d'objet, propriétés de type de données, restrictions de cardinalité…) vers les éléments de la technologie XML Schema (types complexes, types simples, élément, attribut, cardinalités d'occurrences…).
2. un processus de traduction d'une requête sémantique formulée avec SPARQL en une requête construite avec les opérateurs algébriques. Cette dernière sera par la suite traduite en une requête supportée par le SGBD XML natif basée sur XQUERY.

Dans le chapitre suivant nous allons implémenter une simulation de transformation du modèle de donnée sémantique OWL-DL vers son homologue XML Schema du niveau syntaxique de la base de données XML native en utilisant le langage de transformation XSLT à base de balise, et qui permet de naviguer facilement dans une arbre XML source et créer des éléments dans l'arbre cible, en vue de montrer la faisabilité de la couche sémantique proposée et la conformité des neuf règles proposées de transformation de l'ontologie OWL-DL vers XML Schema.

Implémentation d'une simulation de transformation du modèle de données sémantique OWL-DL vers XML Schéma

1. Introduction

Dans le chapitre précédent, nous avons proposé une architecture du modèle sémantique d'une base de données basée sur la combinaison entre : le langage d'ontologie OWL-DL et les bases de données XML natives. Cette architecture est basée sur l'ajout d'une couche sémantique au dessus du SGBD XML natif.

La couche sémantique de cette architecture est constituée de deux composants :

1. le module des connaissances sous le langage d'ontologie OWL-DL.
2. le module de la prise en charge des requêtes (de recherche) sémantique sous le langage de requête sémantique SPARQL.

En pratique, l'architecture proposée peut dégager deux axes d'implémentation :

3. un processus de transformation des éléments du langage OWL-DL (classes, propriétés d'objet, propriétés de type de données, restrictions de cardinalité…) vers les éléments de la technologie XML Schema (types complexes, types simples, élément, attribut, cardinalités d'occurrences…).
4. un processus de traduction d'une requête sémantique formulée avec SPARQL en une requête construite avec les opérateurs algébriques. Cette dernière sera par la suite traduite en une requête supportée par le SGBD XML natif basée sur XQUERY.

Nous exposons dans ce chapitre l'implémentation de la simulation de transformation du modèle de donnée sémantique OWL-DL vers son homologue XML Schema du niveau syntaxique de la base de données XML native. Le langage de transformation XSLT sur la plateforme oXygen[11] est choisi, pour concrétiser ce mécanisme de transformation des différents concepts de l'ontologie vers les différents éléments appropriés du schéma XML

[11] http://www.oxygenxml.com/.

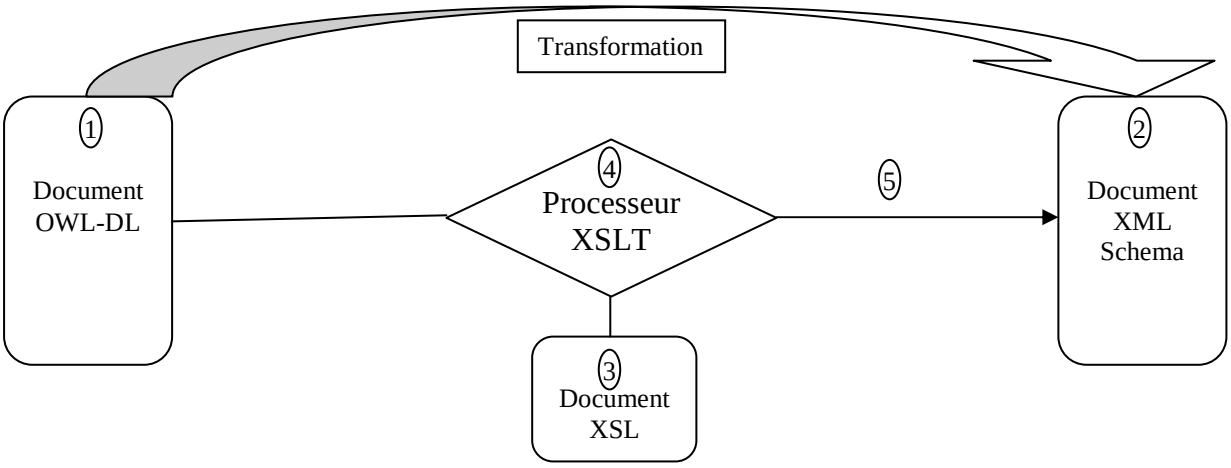
cible de la base de données, puisque la syntaxe de la structure du langage OWL est à base XML (structure de balise), ceci permettra de localiser les balises sur lesquelles on établit des conditions, et enfin choisir le format cible.

L'objectif est de montrer la faisabilité de la couche sémantique proposée et la conformité des neufs règles de transformation de l'ontologie OWL vers XML Schema proposées dans le chapitre précédent.

2. Etapes à suivre pour la simulation

Nous exposons ici un ensemble d'étapes de l'implémentation de la simulation de transformation des différents composants du modèle de donnée sémantique OWL-DL vers les éléments XML Schema du niveau syntaxique de la base de données XML native, en utilisant le langage de transformation XSLT.

Le langage de transformation XSLT est choisi, pour concrétiser le mécanisme de transformation des différents concepts de l'ontologie vers les différents éléments appropriés du schéma XML de la base de données cible (Fig 4.1.), car la syntaxe de la structure du langage OWL est à base XML (structure de balise). Ce dernier permettra de localiser les balises sur lesquelles on établit des conditions, et enfin choisir le format cible qui est sous la syntaxe du XML Schema.



1 : le document source de l'ontologie avec l'extension .owl.
2 : le document résultat de l'opération de transformation avec l'extension .xsd XML Schema).
3 : le document avec l'extension .xsl comportant les différentes règles de modèle assurant la création du fichier XML Schéma résultat.
4 : le processeur de transformation comportant le scénario de déroulement de l'opération.
5 : le sens de transformation.

Fig 4.1. Scénario de transformation

En outre, d'autres outils sont exploités ici pour mettre en évidence l'implémentation de la simulation, à savoir l'éditeur protégé, qui fournit une interface modulaire, permettant à l'utilisateur d'éditer, de visualiser et d'enregistrer de l'ontologie ; et l'éditeur oXygen XML Editor qui offre un moyen d'éditer et de visualiser d'arbres XML, ainsi qu'un debugger du langage de transformation XSLT.

La démarche à suivre pour la simulation comporte les étapes suivantes :

1. Editer l'ontologie OWL-DL exemple par l'outil Protégé.
2. Copie et afficher l'ontologie OWL-DL par l'outil oXygen XML Editor.
3. Construire le processeur XSLT, en appliquant les règles de transformation mentionnées en chapitre III.

2.1. Edition de l'ontologie OWL-DL sous Protégé

Afin d'avoir une ontologie OWL-DL correcte syntaxiquement, dans cette étape, nous proposons l'utilisation de l'outil, qui est fondé sur le modèle de la logique de description, Protégé version 3.1.1. Cet outil fournit une interface modulaire, permettant à l'utilisateur d'éditer, de visualiser et d'enregistrer de l'ontologie au format OWL en générant deux types de fichiers : Un fichier projet Protégé (.pprj) et un fichier OWL contenant le code généré OWL (.owl). Protégé est un outil développé en Java et qui nécessite pour fonctionner la machine virtuelle Java (environnement de Java).

L'interface de Protégé est assez simple, l'ensemble des fonctionnalités de l'éditeur étant regroupé en cinq onglets. Le premier onglet présente les classes de l'ontologie. Il est possible de créer, modifier, supprimer une classe, et de lui attacher des propriétés. Ces propriétés peuvent elles-même être caractérisées. La figure (fig 4.2.) illustre l'écran principal de Protégé.

Nous présentons dans ce qui suit les principales actions à suivre pour réaliser le fichier OWL-DL de l'ontologie de notre exemple courrier.owl en utilisant l'outil Protégé.

2.1.1. Création d'un nouveau projet OWL

Pour créer un nouveau projet OWL, nous devons cliquer sur le bouton file/new project du menu de Protégé et choisir l'option OWL files (fig 4.3.).

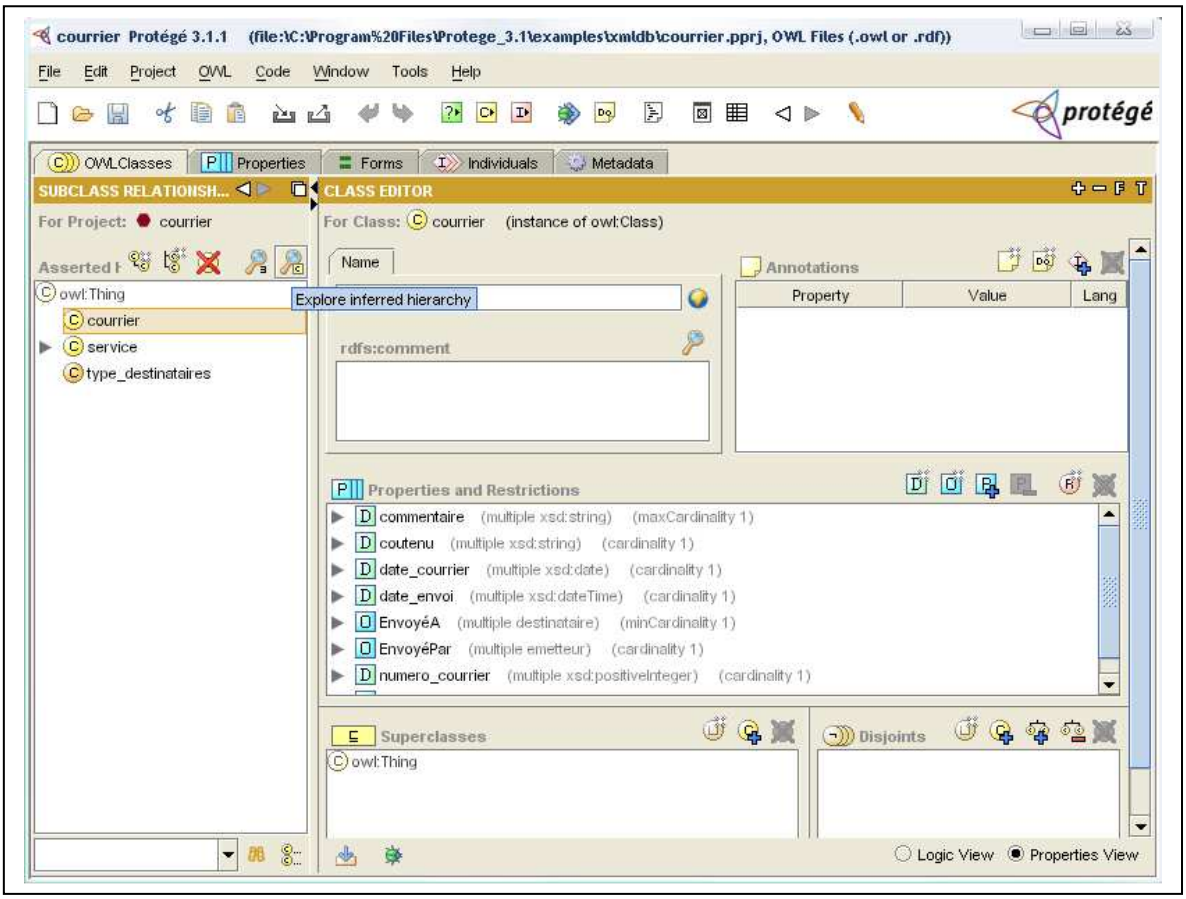


Fig 4.2. Ecran principal de Protégé

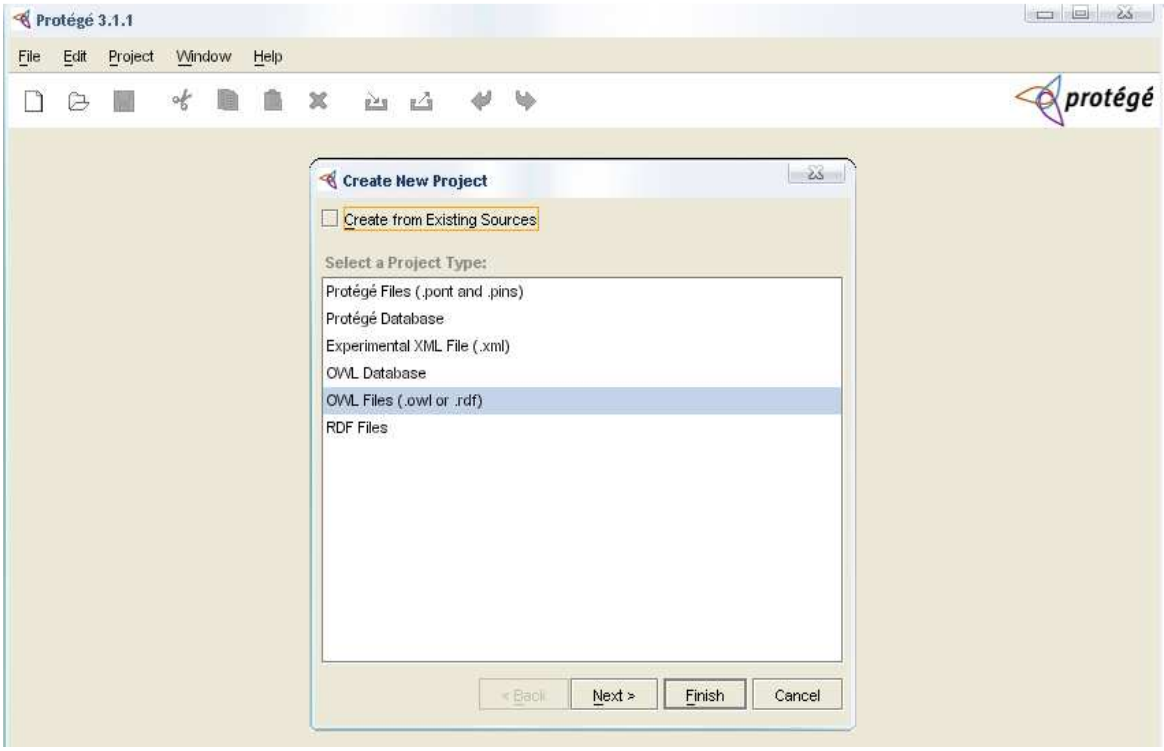


Fig 4.3. Création d'un nouveau projet

2.1.2. Définition des classes et hiérarchie des classes

Cette phase consiste à développer la hiérarchie des concepts (classes) de l'ontologie courrier de notre exemple (fig 4.4)

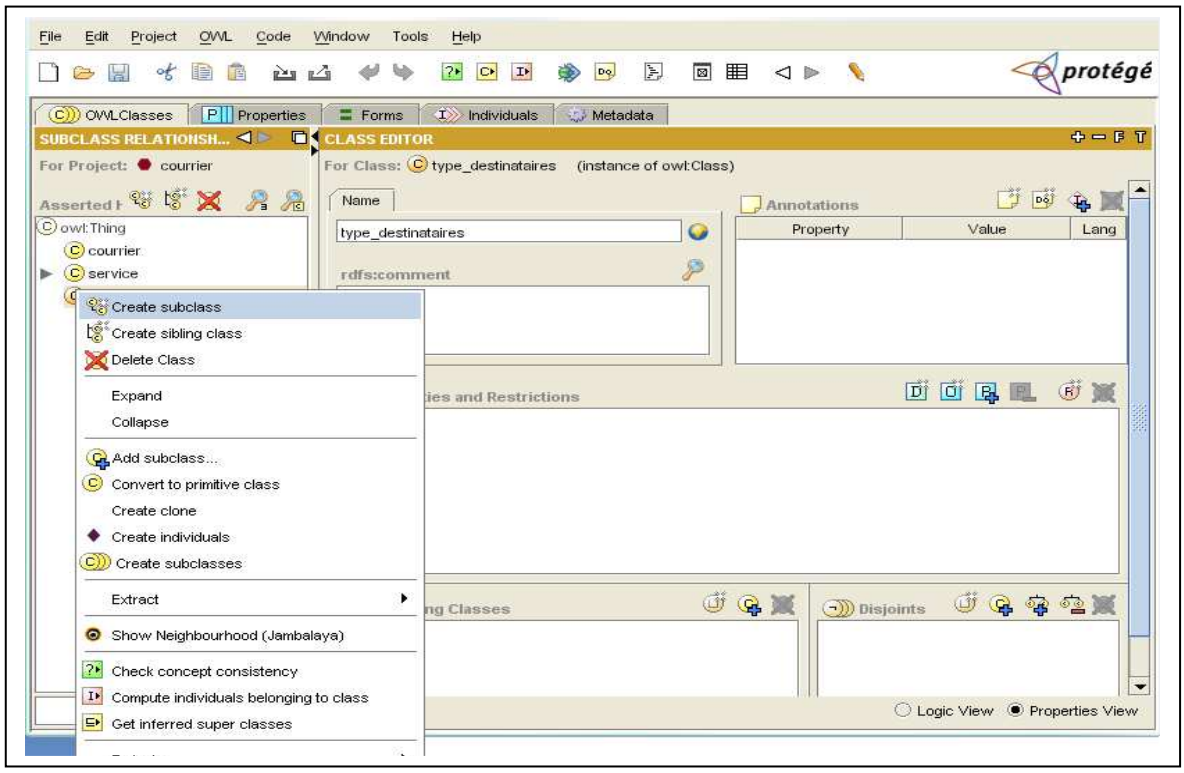


Fig 4.4. Création de la hiérarchie des classes

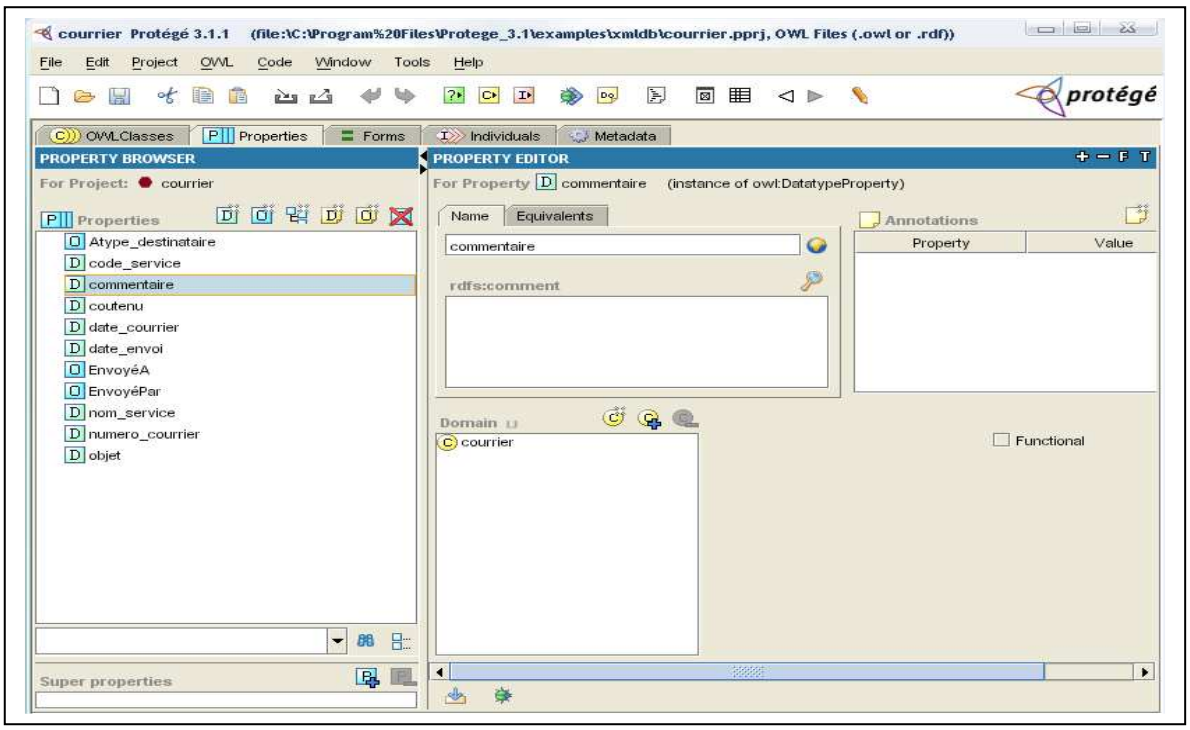


Fig 4.5. Création des propriétés reliant les classes

2.1.3. Définition des propriétés des classes

Les classes seules ne fourniront pas assez d'information. Après avoir défini les classes de notre exemple courrier, nous devons décrire la structure interne de ces concepts par la définition des propriétés d'objet et les propriétés types de données de ces classes. Pour chaque propriété dans la liste nous devons déterminer la classe qu'elle décrit (fig 4.5).

2.1.4. Définition des cardinalités des propriétés des classes

La cardinalité des attributs définit le nombre de valeurs qu'une propriété peut avoir. Protégé permet de spécifier une cardinalité minimale et maximale pour décrire plus précisément le nombre de valeurs d'une propriété (fig 4.6).

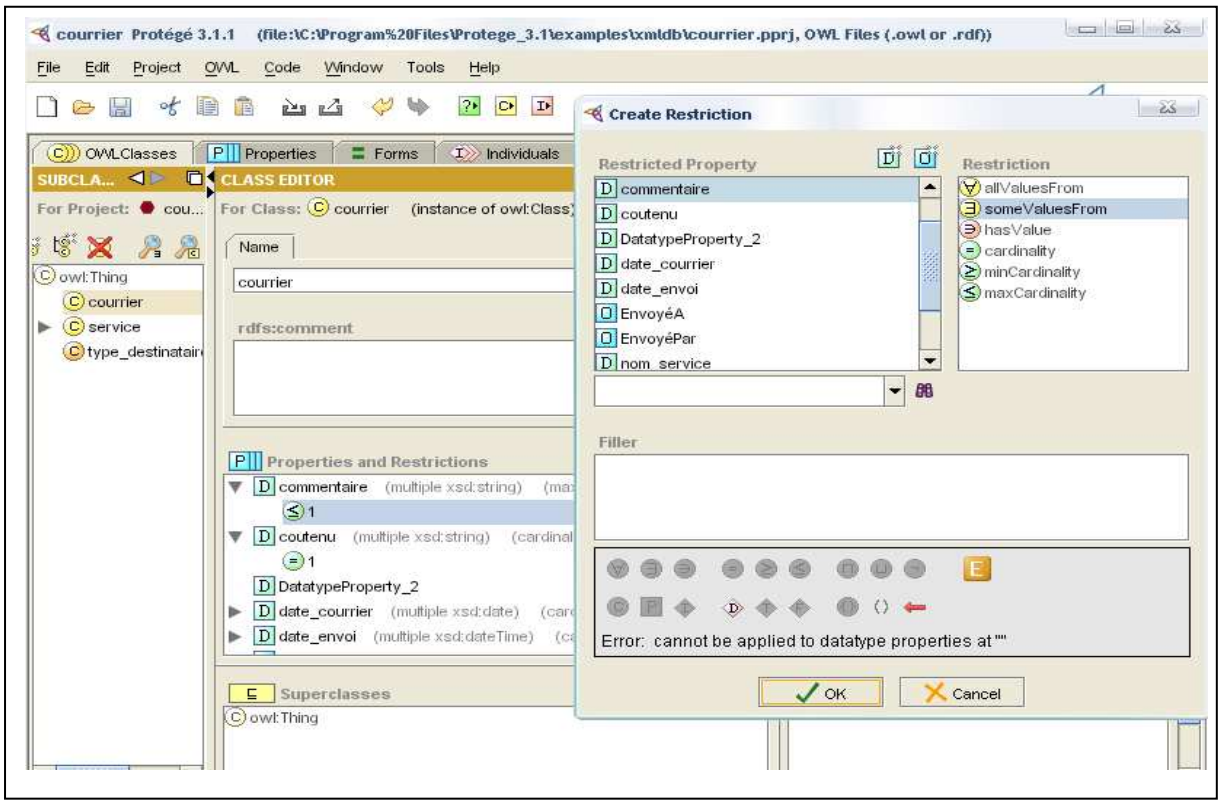


Fig 4.6. Spécifier les cardinalités des propriétés des classes

2.1.5. Affichage de code source de l'ontologie

A l'issue de la création de tous les concepts de l'ontologie par l'assistant, nous pouvons afficher le code source de l'ontologie courrier.owl en appuyant sur le bouton show source code de la barre de menu (fig 4.7).

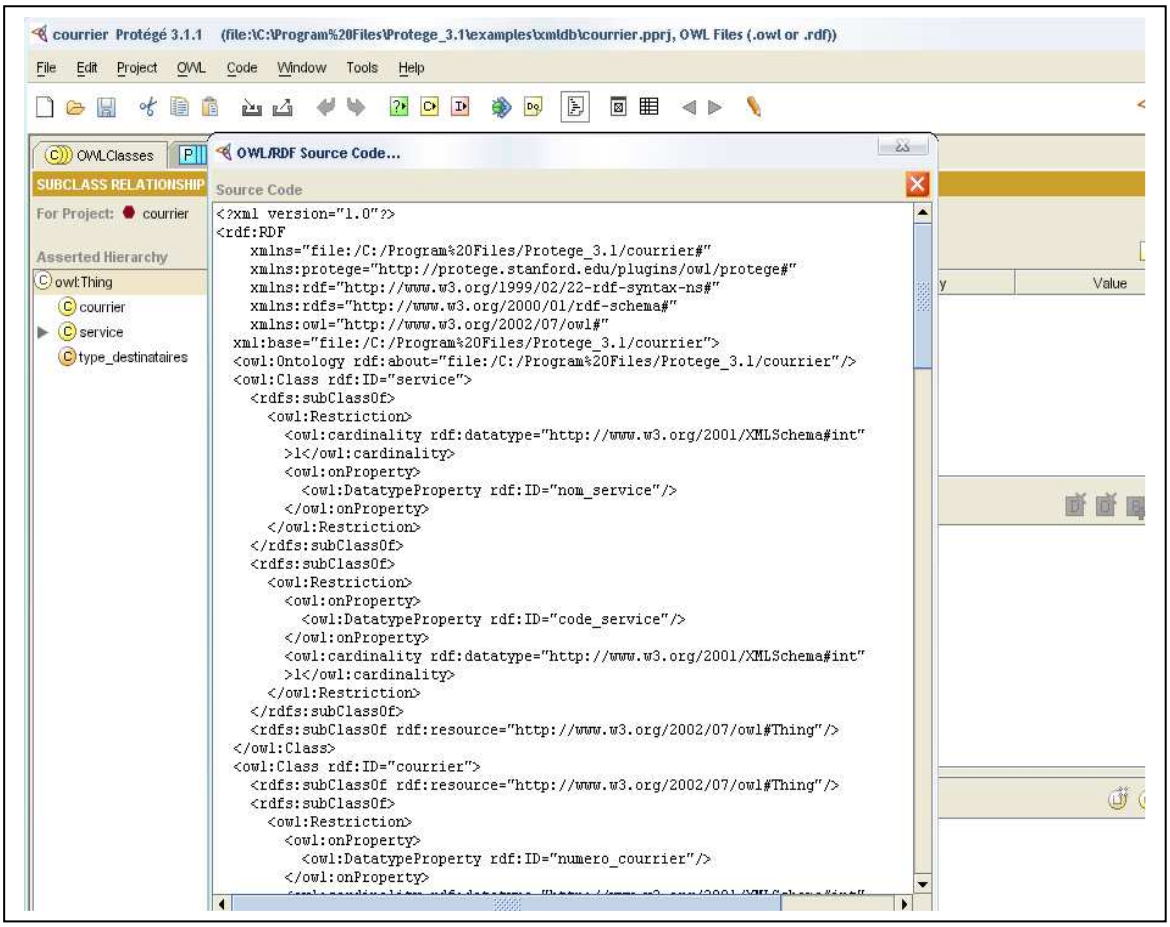


Fig 4.7. Afficher le code source de l'ontologie

2.2. Affichage de l'ontologie OWL-DL par l'outil oXygen XML Editor

Nous allons dans cette simulation utiliser le logiciel oXygen XML Editor version 10. Il s'agit d'un logiciel offrant un certain nombre de fonctionnalités facilitant la saisie de code de langage XML et les langages des technologies liées au XML.

oXygen est un éditeur facile à utiliser qui supporte les dernières technologies et standards XML. Ses fonctions d'aide à l'édition, de débogage XSLT et la possibilité de produire des documents d'autres formats utile à la fois dans les entreprises et dans le milieu universitaire.

La fenêtre principale se divise en deux parties : à gauche, une barre latérale présentant le projet en cours et une exploration du fichier XML en cours d'édition. A droite, et occupant la partie majeure de l'espace disponible, la fenêtre d'édition. Les boutons présents sont habituels, à l'exception d'un bouton propre à l'édition XML. Ce bouton permet d'éditer l'arbre XML.

Cet outil est doté d'un certain nombre de fonctionnalités facilitant la production de documents XML bien formés et valides :

- une édition possible à partir de l'arborescence ;
- une insertion des éléments en accord avec la DTD ou XML Schema.
- une indentation automatique.
- la possibilité de travailler sur des projets.

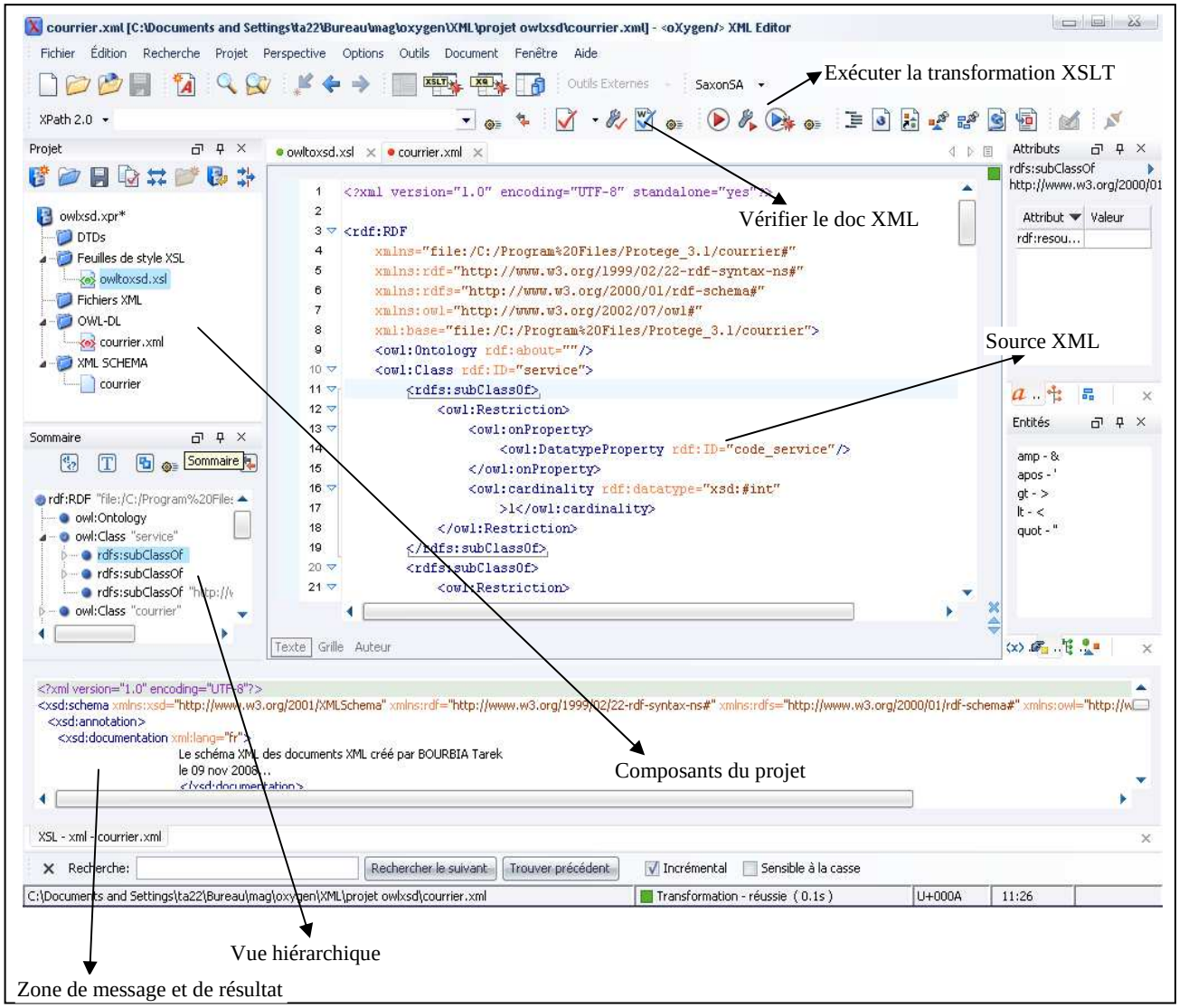


Fig 4.8. Fenêtre principale de l'outil oXygen XML Editor

Pour afficher l'ontologie OWL-DL en tant qu'un fichier XML par l'outil oXygen XML Editor, nous avons deux méthodes :

- Copier le contenu du code source de l'ontologie OWL-DL courier.owl à partir de l'outil Protégé et le copier dans un nouveau fichier XML du répertoire créé auparavant OWL-DL par l'outil oXygen XML Editor.
- Enregistrer l'ontologie OWL-DL courier.owl en tant que fichier dont l'extension .owl par l'outil Protégé et l'ouvrir par l'outil oXygen XML Editor en tant qu'un fichier XML.

2.3. Construction du processeur de transformation XSLT

Le langage de transformation XSLT est utilisé, pour concrétiser le mécanisme de transformation des différents concepts de l'ontologie vers les différents éléments appropriés du schéma XML de la base de données cible, car la syntaxe de la structure du langage OWL est à base XML. Ceci permettra de localiser les balises sur lesquelles on établit des conditions, et enfin choisir le format cible qui est sous la syntaxe du XML Schema.

L'outil oXygen XML Editor dispose d'un debugger XSLT. Alors, nous allons exploiter les facilités offertes par l'interface de cet outil pour créer le processeur XSLT assurant la transformation de l'ontologie OWL-DL vers XML Schéma.

La transformation XSLT décrit les règles pour transformer un arbre XML source en un arbre XML résultat. La transformation est obtenue en appliquant un ensemble de règles de modèles. Une règle modèle est constituée de deux parties : un motif qui sert à identifier des nœuds de l'arbre source et un modèle pouvant être instancié afin de construire une partie de l'arbre résultat.

La structure de l'arbre résultat peut être complètement différente de la structure de l'arbre source comme dans le cas notre simulation, l'arbre source est une ontologie OWL-DL et l'arbre résultat est un schéma XML. Un modèle de règle peut accéder et récupérer les valeurs de chaînes de caractères situées à n'importe quel endroit de l'arbre source; il peut générer des structures répétées au fur et à mesure que les occurrences d'éléments sont rencontrées dans l'arbre source.

XSLT utilise le langage de navigation et d'expression XPATH pour :

- Sélection des éléments pour traitement.

- Spécification des conditions pour les différentes manières de traitement d'un élément.
- Génération des éléments à insérer dans l'arbre résultat.

Avant de détailler les règles modèles (motif de l'arbre source et modèles crées dans l'arbre résultat) assurant la transformation d'une ontologie OWL-DL vers XML Schéma, nous allons associer un scénario de transformation au processeur XSLT à éditer. Le scénario comprend trois parties (fig 4.9) :

- L'entrée XSLT qui présente ici la structure en arbre du document OWL-DL.
- Le nom du processeur XSLT.
- L'arbre en sortie qui présente le document XML Schéma.

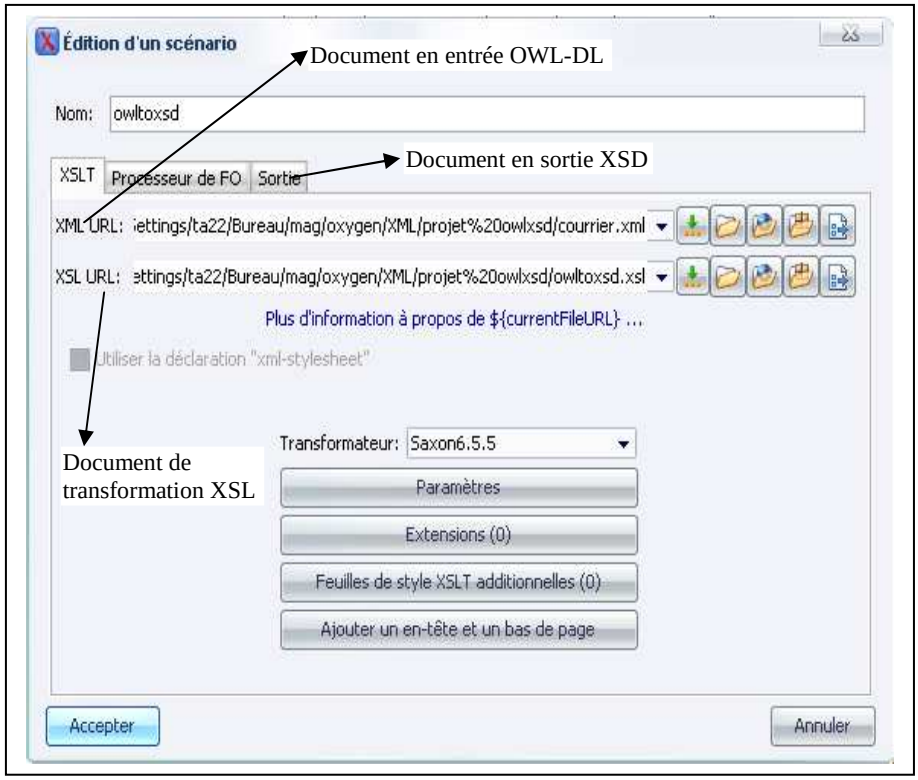


Fig 4.9. Editer un scénario pour appliquer le processeur XSLT

2.3.1. Entête du document de transformation

Le plus important dans l'entête du document de transformation XST, est de spécifier le type de document résultat, ici est un fichier XML Schéma c'est-à-dire un fichier XML, et de préciser les espaces de noms qui vont être dans le document résultat (fig 4.10).

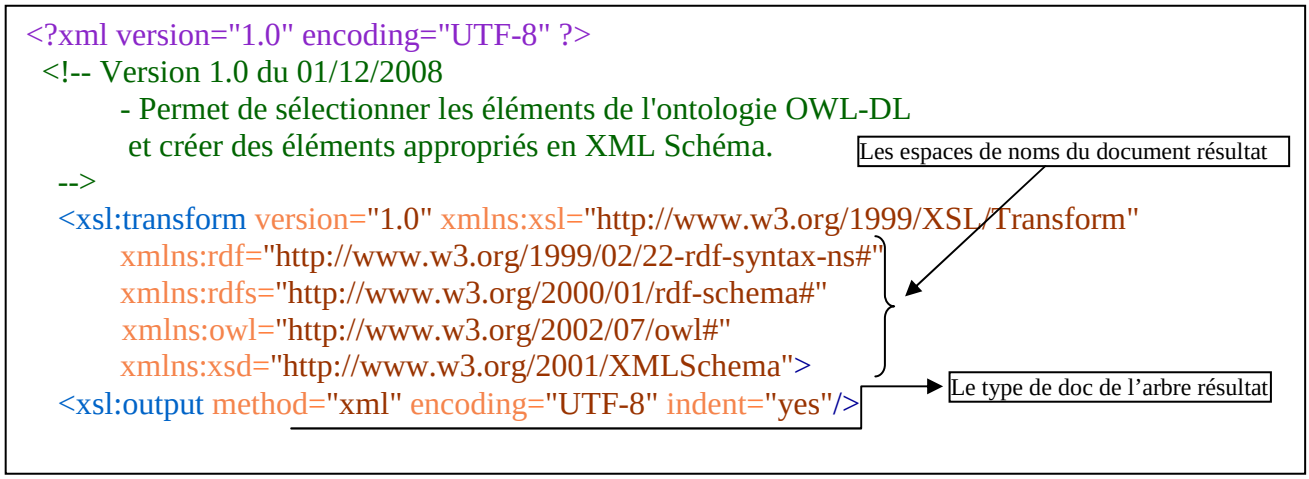


Fig 4.10. Entête du document XSLT

2.3.2. Règle modèle de l'annotation

La règle modèle de l'annotation permet d'une part de créer l'élément racine de l'arbre résultat (<xsd:schema>) qui va contenir les éléments transformés, et d'autre par d'introduire l'annotation XML Schema dont l'objectif est de commenter le schéma de manière à ce que les commentaires puissent être lus aussi bien par des humains que par des applications (fig 4.11).

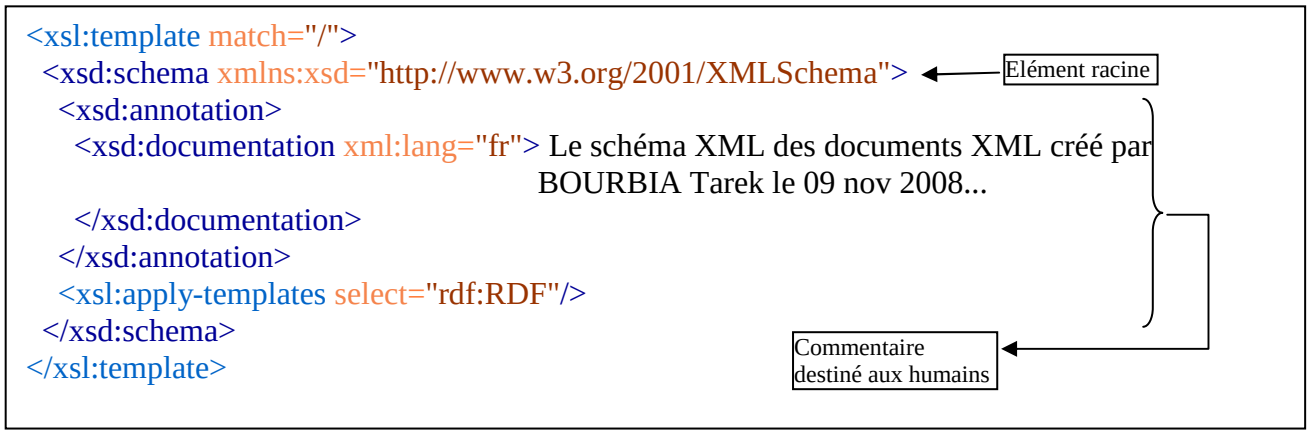


Fig 4.11. Annotation de l'arbre résultat

2.3.3. Règle modèle des classes

Les classes du langage OWL (owl :class) seront transformées vers le type complexe (xsd :complexType) du langage XML Schéma (fig 4.12), à condition que la classe soit, au moins, un domaine de propriétés (rdfs :domain) du propriété de type de donnée (owl :DatatypeProperty).

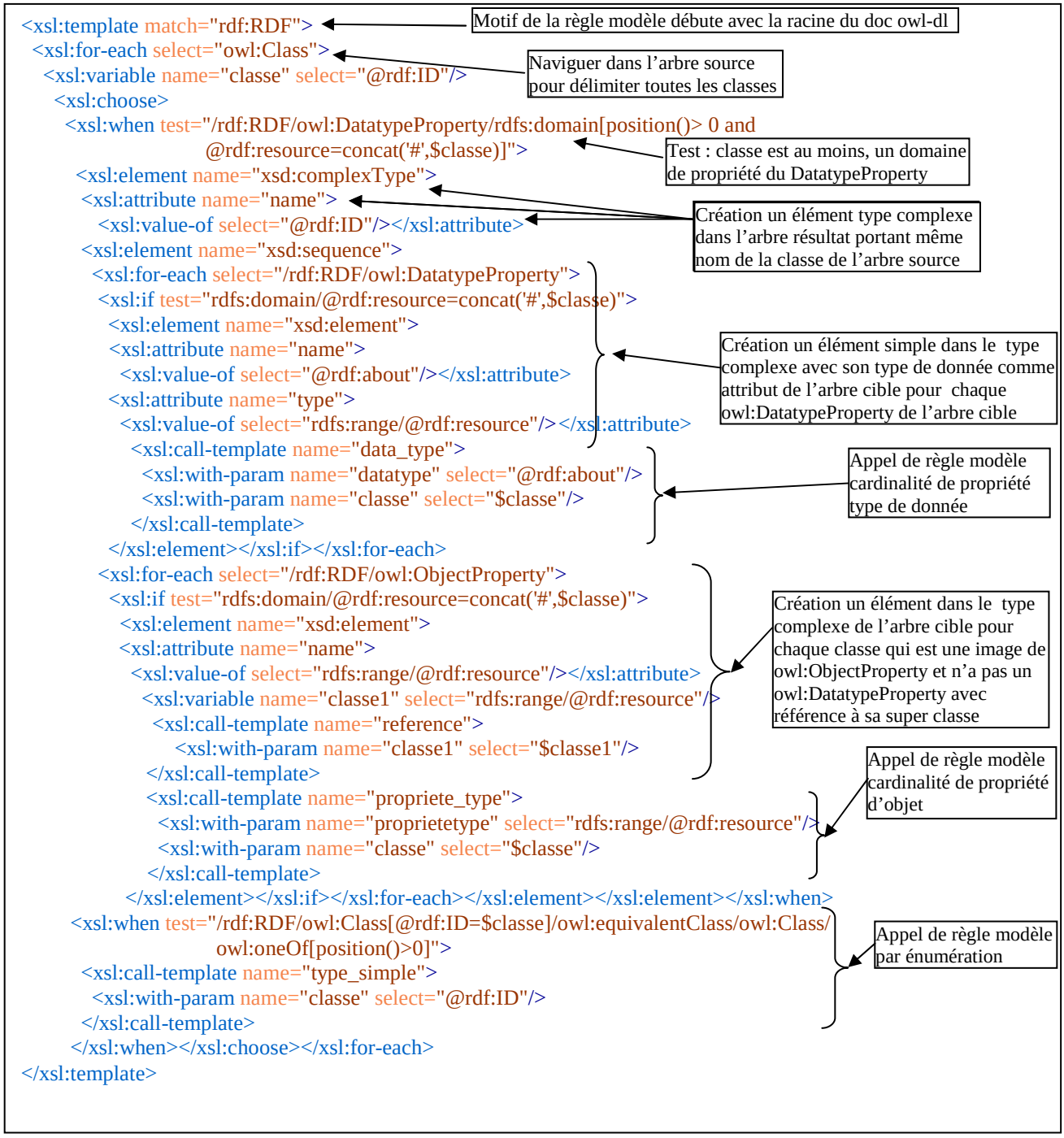


Fig 4.12. Règle modèle des classes OWL-DL

2.3.4. Eléments du type complexe

Toute propriété de type de donnée (owl :DatatypeProperty) de l'arbre source de l'ontologie où le domaine de définition de propriété (rdfs :domain) est la classe transformée à un type complexe (xsd :complexType), sera une déclaration d'élément simple (xsd :element) de ce type complexe de l'arbre résultat (fig 4.12).

Le langage OWL-DL utilise la plupart des types de données intégrés du schéma XML. Ceci facilite la conservation des types de données des différents éléments transférés de l'arbre source OWL-DL vers l'arbre résultat XML Schéma (fig 4.12).

Le classe, de l'arbre source de l'ontologie OWL-DL, qui n'est pas transformée vers un type complexe de XML Schema de l'arbre résultat, parce qu'elle ne figure pas comme un domaine de propriétés (rdfs :domain) d'une propriété (owl :DatatypeProperty) de l'arbre source, sera traitée comme suit : s'il existe une propriété d'objet (owl :ObjectProperty) dans l'arbre source de l'ontologie où le domaine de définition de propriété (rdfs :domain) est une classe transformée à un type complexe (xsd :complexType), l'image de la propriété d'objet (rdfs :range) sera une déclaration d'élément simple (xsd :element) du type complexe de l'arbre résultat de la classe déjà transformée (fig 4.12).

2.3.5. Règle modèle des références

La classe, de l'arbre source de l'ontologie OWL-DL, qui n'est pas transformée vers un type complexe de XML Schema de l'arbre résultat, parce qu'elle ne figure pas comme un domaine de propriétés (rdfs :domain) d'une propriété (owl :DatatypeProperty) de l'arbre source, sera un élément de type complexe avec comme type une référence à sa super classe (fig 4.12) et (fig 4.13).

```
<xsl:template name="reference">
  <xsl:param name="classe1"/>
    <xsl:if test="/rdf:RDF/owl:Class[concat('#',@rdf:ID)=$classe1]/
                  rdfs:subClassOf/@rdf:resource">
      <xsl:attribute name="ref">
        <xsl:value-of select="/rdf:RDF/owl:Class[concat('#',@rdf:ID)=$classe1]/
                              rdfs:subClassOf/@rdf:resource"/>
      </xsl:attribute>
    </xsl:if>
</xsl:template>
```

Création du type de cet élément comme un attribut portant la référence de la classe supérieure

Fig 4.13. Référence de l'élément créé

2.3.6. Règle modèle de classe par énumération

Les classes, de l'arbre source de l'ontologie OWL-DL, définies par énumération de ses instances seront transformées vers le type simple (xsd:simpleType) de l'arbre résultat avec énumération de chaque valeur (fig 4.12) et (fig 4.14).

```
<xsl:template name="type_simple" match="rdf:RDF/owl:Class">
 <xsl:param name="classe"/>
 <xsl:element name="xsd:simpleType">
  <xsl:attribute name="name">
   <xsl:value-of select="$classe"/>
  </xsl:attribute>
 <xsl:element name="xsd:restriction">
  <xsl:attribute name="base">xsd:string</xsl:attribute>
   <xsl:for-each select="self::owl:Class[@rdf:ID=$classe]/owl:equivalentClass/
                         owl:Class/owl:oneOf/*">
    <xsl:element name="xsd:enumeration">
     <xsl:attribute name="value">
      <xsl:value-of select="@rdf:ID"/>
     </xsl:attribute></xsl:element></xsl:for-each></xsl:element></xsl:element>
</xsl:template>
```

Création d'un type simple dans l'arbre cible

Enumération des valeurs du type simple par itération

Fig 4.14. Règle modèle des classes OWL-DL définies par énumération

2.3.7. Règle modèle de cardinalité de propriété type de donnée

Les contraintes de cardinalités (qui permettent d'exprimer le nombre d'éléments dans une relation) de l'arbre source de l'ontologie OWL-DL (owl:mincardinality, owl:maxcardinality et owl:cardinality) seront transformées vers l'arbre cible XML Schema comme des attributs des contraintes d'arité (xsd:minOccurs et xsd:maxOccurs), qui permettent de déterminer le nombre d'occurrences d'un élément dans un document XML (fig 4.12) et (fig 4.15).

L'élément est défini optionnel par le biais de la valeur 0 de l'attribut minOccurs. En général, un élément est requis quand la valeur de minOccurs vaut 1 ou plus. Le nombre maximum de fois qu'un élément peut apparaître est déterminé dans sa déclaration par la valeur de l'attribut maxOccurs. Cette valeur peut être un entier positif, ou encore le mot unbounded qui signifie qu'il n'y a pas de valeur limite. La valeur par défaut dans les deux cas (minOccurs et maxOccurs) est 1. Si les deux attributs (minOccurs et maxOccurs) sont omis, l'élément doit apparaître exactement une seule fois.

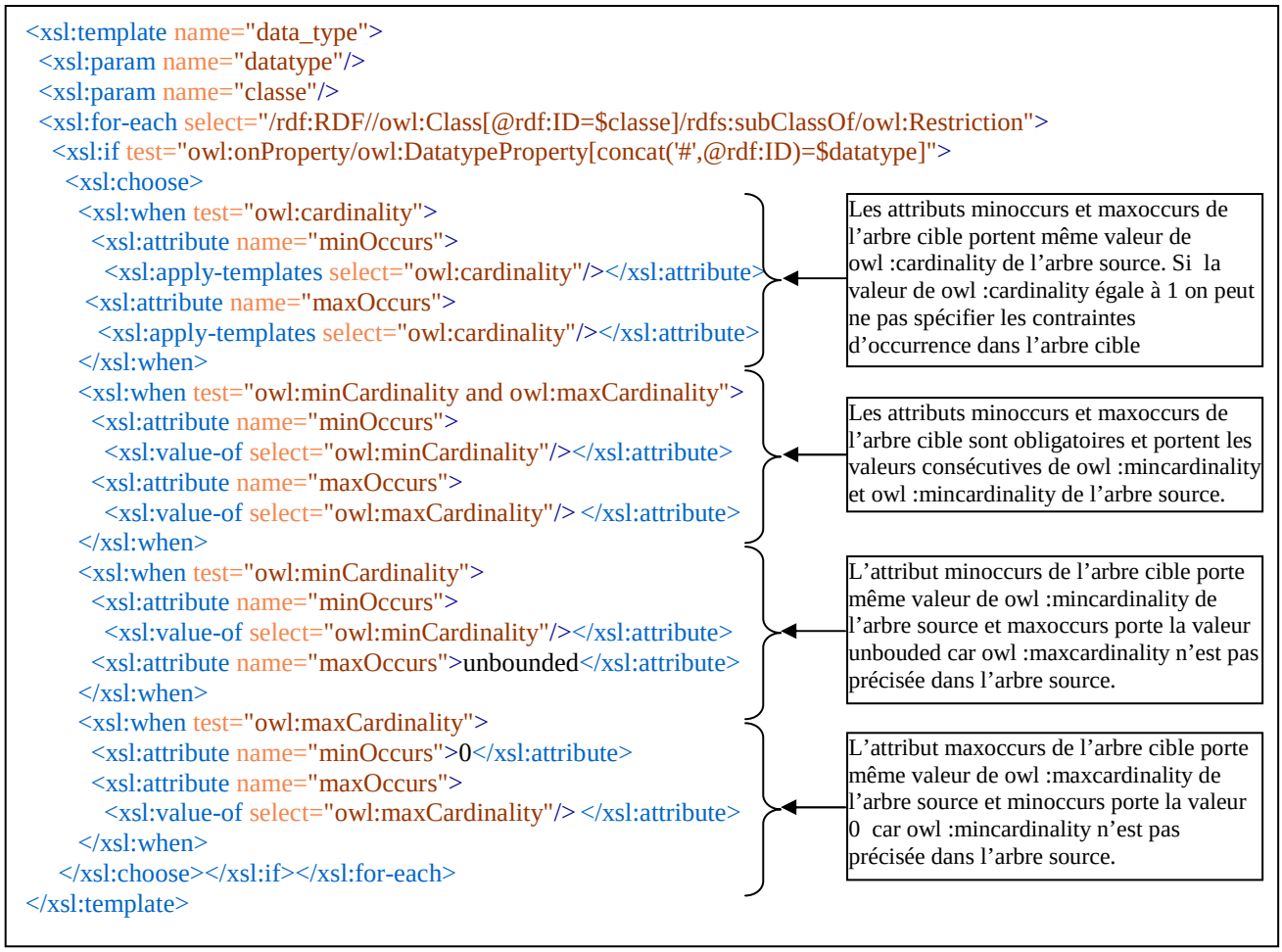


Fig 4.15. Règle modèle de cardinalité de propriété type de données

2.3.8. Règle modèle de cardinalité de propriété d'objet

Les contraintes de cardinalités de propriété d'objet de l'arbre source de l'ontologie OWL-DL (owl:mincardinality, owl:maxcardinality et owl:cardinality) seront transformées vers l'arbre cible XML Schema comme des attributs des contraintes d'arité (xsd:minOccurs et xsd:maxOccurs) exactement comme celles de propriétés type de données (fig 4.12) et (fig 4.16).

```
<xsl:template name="propriete_type">
 <xsl:param name="proprietetype"/>
 <xsl:param name="classe"/>
 <xsl:for-each select="/rdf:RDF/owl:ObjectProperty">
  <xsl:if test="rdfs:domain[@rdf:resource=concat('#',$classe)] and rdfs:range[@rdf:resource=$proprietetype]">
   <xsl:variable name="proprieteobjet" select="@rdf:about"/>
   <xsl:for-each select="/rdf:RDF//owl:Class[@rdf:ID=$classe]/rdfs:subClassOf/owl:Restriction">
    <xsl:if test="owl:onProperty/owl:ObjectProperty[concat('#',@rdf:ID)=$proprieteobjet]">
     <xsl:choose>
      <xsl:when test="owl:cardinality">
       <xsl:attribute name="minOccurs">
       <xsl:apply-templates select="owl:cardinality"/></xsl:attribute>
       <xsl:attribute name="maxOccurs">
       <xsl:apply-templates select="owl:cardinality"/></xsl:attribute>
      </xsl:when>
      <xsl:when test="owl:maxCardinality">
       <xsl:attribute name="minOccurs">0</xsl:attribute>
       <xsl:attribute name="maxOccurs">
       <xsl:value-of select="owl:maxCardinality"/></xsl:attribute>
      </xsl:when>
      <xsl:when test="owl:minCardinality">
       <xsl:attribute name="minOccurs">
       <xsl:value-of select="owl:minCardinality"/></xsl:attribute>
       <xsl:attribute name="maxOccurs">unbounded</xsl:attribute>
      </xsl:when>
      <xsl:when test="owl:minCardinality and owl:maxCardinality">
       <xsl:attribute name="minOccurs">
       <xsl:value-of select="owl:minCardinality"/></xsl:attribute>
       <xsl:attribute name="maxOccurs">
       <xsl:value-of select="owl:maxCardinality"/></xsl:attribute>
      </xsl:when>
     </xsl:choose>
 </xsl:if></xsl:for-each></xsl:if></xsl:for-each>
</xsl:template>
```

Fig 4.16. Règle modèle de cardinalité de propriété d'objet

3. Conclusion

Dans ce chapitre nous avons concrétisé par une simulation d'implémentation une partie de couche sémantique à base du langage d'ontologie OWL-DL

Ce système souffre de quelques insuffisances telles que :

- L'absence d'un algorithme d'interrogation des ontologies.
- Il ne décrit pas la partie qui concerne la combinaison des résultats obtenus en une seule réponse destinée à l'utilisateur.

Nous nous sommes intéressés plus particulièrement à la transformation des différents concepts de l'ontologie OWL-DL vers les différents éléments appropriés du XML Schema. Nous envisageons de raffiner le mécanisme de transformations à partir de la syntaxe et de la sémantique des requêtes exprimées par le langage de requête sémantique SPARQL vers le langage de manipulation des bases données XML natives XQUERY et XPATH. Aussi, nous envisageons d'implémenter une application avec une interface graphique qui représente le niveau sémantique caché en interaction avec le niveau syntaxique du SGBD XML natif.

Conclusion générale

L'intégration de la sémantique dans le domaine des bases de données est devenue un sujet de recherche ces dernières années. Nous avons présenté dans ce mémoire une architecture de modélisation pour étendre le SGBD XML natif par une couche sémantique à base du langage d'ontologie OWL-DL. La couche sémantique de cette modélisation est composée de deux modules : le module qui concerne la modélisation des connaissances sous le langage d'ontologie OWL-DL et le module de la prise en charge des requêtes sémantiques sous le langage de requête sémantique SPARQL.

L'architecture proposée contient un niveau sémantique au dessus d'un niveau syntaxique, et la correspondance entre ces deux niveaux est assurée par:

1. un processus de transformation des éléments du langage OWL-DL (classes, propriétés d'objet, propriétés de type de données, restrictions de cardinalité…) vers les éléments de la technologie XML Schema (types complexes, types simples, élément, attribut, cardinalités d'occurrences…).

2. un processus de traduction d'une requête sémantique formulée avec SPARQL en une expression algébrique. Cette dernière sera par la suite traduite en une requête XQUERY supportée par le SGBD XML natif.

Nous avons utilisé plusieurs technologies pour construire cette architecture et assurer la correspondance entre le niveau sémantique et le niveau syntaxique, à savoir :

- Le langage de structuration à base des balises XML.
- La description formelle et organisationnelle XML Schema.
- Le langage de transformation XSLT d'un document XML vers un autre document à base XML.
- Le langage de navigation XPATH qui permet de naviguer dans les éléments, les attributs et les textes d'un document XML et de l'interroger.

- Le langage de requête XQuery qui permet l'extraction et la recherche des parties de documents XML.
- Les bases de données XML natives.
- Les langages sémantiques et ontologiques RDF, RDFS et OWL.
- Le langage de requête sémantique SPARQL.

En conclusion ce travail nous a permis d'avoir :

- Une conception et une modélisation logique sous le langage OWL, et un stockage physique sous une base de données XML native.
- Une exploitation sémantique des bases de données offerte par le langage d'ontologie OWL.
- Un mécanisme d'inférence dans le contexte des bases de données.
- Une richesse de fonctionnalités de gestion et manipulation de données à base ontologique offerte par le SGBD XML natif (indexation, optimisation, transaction, sécurité et confidentialité des données…).

Notre travail offre de nombreuses perspectives, à savoir :

- Etendre l'architecture proposée par d'autres modules : module de requête d'insertion et de mise à jour, module de sécurité, module d'importation des données à base ontologique.
- Concevoir deux niveaux d'optimisation des requêtes sémantiques et syntaxiques.
- Développer une plateforme spécifique pour assurer la persistance des différents objets et instances d'objets de l'ontologie OWL sans passer par un processus de transformation.
- Arriver à la notion base de données OWL native.

Références

[1] M.K.Smith, C.Welty, D.L.MCGuinness. "OWL Web Ontology Language", http://www.w3.org/TR/2004/REC-owl-guide-20040210/, 2004.

[2] T. Bray, J. Paoli, C. M. Sperberg-McQueen, E.Maler, Sun Microsystems, F.Yergeau. "Extensible Markup Language (XML) Version 1.0. forth edition", http://www.w3.org/TR/2006/REC-xml11-20060816/, 2006.

[3] B.Verhaegen. "Modélisation et interrogation de données multidimensionnelles en XML", http:// www.code.ulb.ac.be, université libre de Bruxelles, 2007.

[4] P.Buch. "Les langages du Web sémantique", http://www.metarisk.inapg.inra.fr, Département Mathématique et Informatique Appliquées, INRA, 2006

[5] AGRIS/CARIS. "Génération du XML PA AGRIS à partir de bases de données locales", http://www.fao.org/docrep/009/ae908f/ae908f00.htm, 2005.

[6] B.Verhaegen. "Requête OLAP sur une base de données XML native", http:// www.code.ulb.ac.be, université libre de Bruxelles, 2006.

[7] S.Boag, D.Chamberlin, D.Florescu, J.Robie, J.Siméon, and M.F Fernandez. "XQUERY 1.0: A Query language for XML", http://www.w3.org/TR/xquery/, 2007

[8] A.Laux, L.Martin. "XUPDATE XML update language. XML :DB working draft :2000" http://www.xmldb.org/xupdate/index.html.

[9] X.Lacot. "Introduction à OWL, un langage XML d'ontologie web", http://www.lacot.org/public/owl/, 2006.

[10] S.Telghamti. "Validation des requêtes de mise à jour dans les bases de données XML natives". Univ Mentouri Constantine,Algérie 2007.

[11] H.Bohring and S.Auer. "Mapping XML to OWL Ontologies", http://www.informatik.uni-leipzig.de/~auer/publication/, Univ of Leipzig,Germany, 2005.

[12] P.V.Biron and A.Malhotra, "XML Schema Part2: Datatypes", http://www.w3.org/TR/xmlschema-2/, 2004.

[13] J.Clark. "XSL Transformation (XSLT)", http://www.w3.org/TR/xslt,1999.

[14] E. Prud'hommeaux, A. Seaborne. "SPARQL Query Language for RDF", http://www.w3.org/TR/2005/WD-rdf-sparql-query-20050721, 2005.

[15] E.Sirin and B.PArsia. "SPARQL-DL Query for OWL-DL", http://clarkparsia.com/files/pdf/sparqldl.pdf, Departement of Computer Science, University of Manchester, UK, 2006.

[16] R.Bourett. "XML et les bases de données", http:/www.rpbourret.com,2005.

[17] G.Klyne, and J.Carroll. "resource description framework ou rdf : Concepts et syntaxe abstraite", http://www.w3.org/TR/rdf-concepts/, 2004.

[18] D.BRICKLEY et B.MC BRIDE. RDF Vocabulary description language 1.0: RDF Schema. W3C, Recommendation, http://www.w3.org/rdf-schema, 2004.

[19] S.Garllati. "Logique de description LOOM", http://www. public.enst-bretagne.fr, 2008.

[20] A.Berglund. “eXtensible Stylesheet Language (XSL)” http://www.w3.org/TR/2006/rec-xsl20061006/, 2006.

[21] J.Clark. S.DeRose, J. Paoli, and C. Michael. “Langage XML XPATH Version 1.0”. http://www.w3.org/TR/Xpath/, 1999.

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

[23] Steve DeRose, Brown University Scholarly Technology Group, Eve Maler, Sun Microsystems, David Orchard, Jamcracker. XML Linking Language (XLink) Version 1.0 W3C Recommendation 27 June 2001.

[24] Eve Maler (ArborText), Steve DeRose (Inso Corp. and Brown University). XML Pointer Language (XPointer) W3C Working Draft 03-March-1998

[25] S.Boag, D.Chamberlin, D.Florescu, J.Robie, J.Siméon, and M.F Fernandez. XQUERY 1.0: A Query language for XML. W3C Recommondation, 2007

[26] H.Mahboubi, K.Aouiche, J.Darmont. Un index de jointure pour les entrepôts de données XML. ERIC, Université Lumière Lyon 2, 9 juillet 2007.

[27] B.Amman. Stockage des données XML. INRIA

[28] G.A.Silber. Introduction à XML, http://www.cri.ensmp.fr/people/silber/xml, CRI/ENSMP, Mines Paris, 2008.

[29] P.Hayes, RDF semantics. W3C recommondation, http://www.w3.org/TR/owl-semantics/, 2004.

[30] R.Cyganiak. A relational algebra for SPARQL. HP Labs Bristol, UK. 28/09/2005.

[31] T.Berners-Lee, L.Masinter, M.McCahill, Naming and Addressing: URIs, URLs, ..., http://www.w3.org/Adressing/, 2005.

[32] T.Berners-Lee, Uniform Resource Identifier (URI): Generic Syntax: W3C recommondation., http://labs.apache.org/webarch/uri/rfc/rfc3986.html, janvier 2005.

[33] G.Pierra, H.Dehainsala, Y.Ait Ameur,L.Belatreche, Ontology based data base: principle and implementation,

http://cat.inist.fr/?aModele=afficheN&cpsidt=16944438, ISI, vol 10,2, pp. 91-115, France, 2005.

[34] H.Dehainsala, Explication de la sémantique dans les bases de données, thése doctorat, université poitier, France, 2007.

[35] P.Myung, L.Ji-Hyun, L.Chun-Hee, L.Jiexi, O.Serres, C.Chin-Wan, ONTOMS : An Efficient and Scalable Ontology Management System , http://hdl.handle.net/10203/1289, 2007.

[36] Z.Pan, J.Heflin, DLDB: Extending Relational Databases to Support Semantic Web Queries, http://database.ittoolbox.com/documents/academic-articles,2004.

[37] G.Perra, Un modèle formel d'ontologie pour l'ingénierie, le commerce électronique et le Web sémantique: Le modèle de dictionnaire sémantique PLIB, http://www.lalic.paris4.sorbonne.fr/stic/octobre/octobre/apr/Pierra.pdf, 2002.

[38] M.Pagels, The DARPA Agent Markup Language (DAML), http://www.daml.org/, 2000.

[39] J. Broekstra, A. Kampman and F. van Harmelen: Sesame: A Generic Architecture for Storing and Querying RDF, http://www.iswc2004.semanticweb.org/demos/, 2002.

[40] L.H.Arnaud, L.H. Philippe, W.Lauren, N.Gavin, R.Jonathan, C.Mike, B.Steve. Document Object Model (DOM) Level 3 Core Specification, http://www.w3.org/TR/2004/REC-DOM-Level-3-Core-20040407/,2004.

[41] A.Gamache, Modèle relationnel, algèbre relationnelle et transposition du conceptuel au relationnel, Université Laval, Québec, http://www.gamache.developpez.com/LivreBD/LivreElectro1Dev.html, 2005.