# Une approche matérialisée basée sur les vues pour l'intégration de documents XML

Houda Ahmad

**Université Joseph Fourier – Grenoble I** **École Doctorale IMA**

# Une approche matérialisée basée sur les vues pour l'intégration de documents XML

## THÈSE

présentée et soutenue publiquement le 26 juin 2009

pour obtenir le grade de

**Docteur de l'Université Joseph Fourier – Grenoble I**

**(Spécialité : Informatique)**

par

Houda Ahmad

**Composition du jury**

| | |
|---|---|
| *Président :* | Laurent TRILLING Professeur, Grenoble |
| *Rapporteurs :* | Yamine AIT AMEUR Professeur, Université de Poitiers<br>André FLORY Professeur, Université Lyon I |
| *Examinateurs :* | Ana SIMONET Maître de Conférences, UPMF Grenoble (Co-directrice)<br>Michel SIMONET Chargé de recherche CNRS (Directeur de thèse) |

**Laboratoire des Techniques de l'Imagerie, de la Modélisation et de la Cognition**
**TIMC-IMAG — UMR 5525**

Mis en page avec la classe thloria.

# Remerciements

*Je dédie cette thèse*
*à mes parents.*

Professeur, Université Poitiers

# Table des matières

# Liste des tableaux

# Table des figures

# Introduction générale

## 0.1 Problématique de l'intégration de données

Avec l'arrivée de l'Internet et du Web le nombre de sources d'information reliées sémantiquement ainsi que le nombre d'utilisateurs potentiels de ces sources ont connu une augmentation exponentielle durant les dix dernières années. Ces sources sont souvent hétérogènes, autrement dit elles utilisent des modèles différents pour la représentation des données, tels que le modèle relationnel, le modèle semi-structuré dans des collections de pages Web, les fichiers textes, etc. Ainsi, dans le domaine médical, on trouve les données du patient sur une multitude de supports : des tables relationnelles dans les bases de données cliniques, des documents XML issus d'appareillages d'analyse (biologique, IRM, radio, etc.), des documents textuels (comptes-rendus d'examens, lettres de sortie d'hospitalisation, ...), etc.

A cette multiplicité de supports et de formats se rajoute l'hétérogénéité sémantique des diverses sources de données. Cette hétérogénéité reflète la diversité des points de vue des concepteurs des divers systèmes. En conséquence, les langages de programmaton ou d'interrogation de ces sources de données sont différents, ce qui pose de sérieux problèmes aux utilisateurs qui cherchent à combiner ou intégrer des informations provenant des différentes sources. Dans ce contexte, les entreprises doivent répondre à deux besoin essentiels : la disponsibilité rapide des informations inter-sources et la découverte des tendances à partir des données accumulées dans l'entreprise au fil du temps. Ces besoins ont conduit les entreprises à proposer une vision plus globale de leurs informations, autrement dit faire inter-opérer les différentes sources de données.

Le processus d'intégration repose généralement sur un mécanisme de vues. Un tel mécanisme permet d'offrir une vison homogène et unifiée des données provenant de sources différentes, autrement dit il permet à l'utilisateur d'utiliser une structure unique pour accèder à des gros volumes de données hétérogènes.

Le langage XML joue actuellement un rôle croissant dans l'échange des données sur le Web. C'est un langage de balisage qui permet de représenter la structure des données semi-structurées. Ces données sont irrégulières et auto-décrites, c'est-à-dire d'une part que des documents similaires peuvent être représentés par des schémas différents, et d'autre part que le schéma de chaque document est auto-contenu dans le document même. C'est pourquoi les algorithmes et les techniques d'intégration et d'interrogation de telles sources de données sont souvent plus complexes que ceux définis pour l'intégration et l'interrogation de sources de données structurées. En conséquence, il est nécessaire de trouver des techniques et des méthodologies permettent d'interroger d'une manière efficace et simple les données XML volumineuses.

La simplicité et la flexibité de XML ont fait qu'il s'est rapidement imposé comme modèle commun pour l'intégration des données. Pour cette raison, la définition d'un mécanisme de vues pour XML est important. Il y a eu plusieurs projets sur les modèles de vues pour XML. Parmi eux on peut citer : Xylème [Xyleme, 2001], VIMIX [Baril, 2003], Xyview [Sebi, 2007]. Ces systèmes seront présentés dans le chapitre 3.

L'utilisation d'un modèle de vues pour XML a deux intérêts : Tout d'abord, il permet d'utiliser XML comme langage commun pour intégrer des données ; l'utilisation de XML comme modèle pivot présente plusieurs avantages, qui seront présentés dans le chapitre 2 (cf. section 2.7). Ensuite, il permet de faciliter l'accès à des documents XML, comme dans le contexte classique des SGBD.

## 0.2 Contribution

La principale contribution de notre travail est un modèle matérialisé basé sur les vues pour l'intégration de documents XML. Ce modèle est appelé OSIX (Osiris-based model for the Integration of XML documents).

L'objectif de ce modèle est l'intégration de données XML en utilisant les principes d'Osiris, un prototype de SGBD-BC, dont le concept central est celui de vue. Dans ce système, une famille d'objets est définie par une hiérarchie de vues, où chaque vue est définie par ses vues mères, ses attributs et contraintes propres. Osiris appartient à la famille des logiques de description, la vue minimale d'une famille d'objets étant assimilée à un concept primitif et ses autres vues à des concepts définis. Un objet d'une famille satisfait certaines de ses vues. Pour chaque famille d'objets, Osiris construit, par analyse des contraintes définies dans toutes ses vues, un espace de classement n-dimensionnel. Cet espace sert de support au classement d'objets et aussi à leur indexation.

Dans cette thèse nous avons étudié l'apport des principales fonctionnalités d'Osiris - classement, indexation et optimisation sémantique des requêtes - à l'intégration de documents XML. Pour cela nous produisons un schéma XML cible (schéma abstrait), qui représente un schéma Osiris ; chaque document satisfaisant un schéma XML source (schéma concret) est réécrit en termes du schéma cible avant de subir l'extraction des valeurs de ses entités. Les objets correspondant à ces entités sont alors classés et indexés. Le mécanisme d'optimisation sémantique des requêtes d'Osiris peut dès lors être utilisé pour extraire les objets d'intérêt pour une requête.

## 0.3 Organisation du mémoire

Dans le chapitre 1, nous présentons un état de l'art sur la notion de données semi-structurées et plus particulièrement les données XML. Nous étudions les formalismes de représentation et les méta-données pour les données semi-structurées. Enfin, nous présentons les langages d'interrogation et les systèmes de gestion de données XML.

Dans le chapitre 2, nous nous focalisons sur la notion de données hétérogènes, leur types et les approches utilisées pour leur intégration : l'approche médiateur et l'approche entrepôt de données. Ensuite, nous passons à l'étude des approches existantes pour le traitement de requêtes. Enfin, le chapitre se termine par une représentation des systèmes d'intégration basés sur XML.

Dans le chapitre 3, nous présentons le concept de vue, qui est utilisé pour l'intégration de données. Nous étudions la définition des vues dans différents modèles de données : le modèle relationnel, le modèle orienté-objet et le modèle semi-structuré ; nous nous concentrons sur le modèle de données semi-structuré, et plus précisément sur les données XML.

Dans le chapitre 4, nous présentons le système Osiris, qui implémente le modèle des P-types, et ses concepts de base. Nous nous sommes intéressés à l'utilisation du concept de P-type dans l'intégration de sources hétérogènes de données XML. Nous nous focalisons sur l'utilisation du concept de vue, qui est central dans le modèle des P-types et qui se révèle être un concept clé pour l'intégration de données XML, et sur l'indexation offerte par Osiris, qui permet l'optimisation sémantique de requêtes.

Dans le chapitre 5, nous présentons notre approche (OSIX : Osiris-based System for the integration of XML documents), conçue pour l'intégration des documents XML. Nous détaillons l'architecture du système OSIX et ses processus. Enfin, le chapitre se termine par un exemple d'application de notre approche dans le domaine médical.

Dans le chapitre 6, nous présentons le processus général que nous avons suivi pour implémenter le système OSIX. Nous présentons l'outil OSIXT (OSIX Tool) qui applique le système OSIX à l'intégration et l'interrogation de documents XML simulant les données d'un hôpital. Il fournit une interface graphique qui permet à l'utilisateur de saisir une requête sur une source XML et donner la réponse appropriée.

Enfin, la conclusion présente un bilan de notre travail et des perspectives qui permettraient de compléter notre approche.

# Première partie

# État de l'art

# Chapitre 1

# Les données semi-structurées XML

## 1.1 Introduction

Dans ce chapitre nous présentons un état de l'art sur la notion de données semi-structurées et plus particulièrement les données XML. Les données XML offrent une grand flexibilité dans le traitement de requêtes (stockage, chargement), les mises à jour et les changements structurels. en raison d'un schéma qui n'est pas fixé à l'avance. Par contre, cette caractéristique a conduit à une certaine complexité en ce qui concerne le stockage de données, la formulation et l'évaluation de requêtes. Pour ces raisons, le développement d'outils efficaces permettant d'exploiter de telles données et d'extraire les informations pertinentes pour l'utilisateur est devenue une nécessité de première importance.

Ce chapitre est organisé comme suit.
- Dans la section 1.2, nous présentons l'évolution des modèles de données utilisés en bases de données.
- Dans la section 1.3, nous étudions des formalismes de représentation des données semi-structurées.
- Par la suite, nous présentons des méta-données pour les données semi-structurées.
- Dans la section 1.5, nous présentons les langages d'interrogation pour XML.
- Enfin, nous présentons les systèmes de gestion de données XML.

## 1.2 Modèles pour la représentation des données hétérogènes

On peut distinguer deux types de données

### 1.2.1 Données structurées

Les données structurées sont des données disposant d'une structure rigide, qui leur permet d'être facilement stockées dans les tables d'une base de données relationnelle. La gestion de données structurées est bien connue et est assurée par les SGBDs. Avant de stocker des données d'une telle catégorie dans une base, l'utilisateur doit définir un schéma, exprimé dans un Langage de Définition de Données (*LDD*). Ce schéma décrit la structure des données et les liens entre les données de diverses tables.

### 1.2.2 Données semi-structurées

Les données XML (pour lesquels un schéma n'est pas obligatoire) sont appelées des données semi-structurées car ce sont des données qui n'ont pas de schéma a priori mais dont le schéma peut-être extrait à partir des données elles-mêmes. Des exemples de telles données sont les documents, ou plus largement l'ensemble des documents XML d'un site Web ainsi que les liens inter-documents permettant de passer de l'un à l'autre. Les modèles semi-structurés peuvent être vus comme une relaxation du modèle relationnel classique, dans lequel on autorise une structure moins rigide et moins homogène des "champs données". Ce modèle de données s'est avéré très utile pour la représentation de familles de documents variés : multimédia, hypertexte, données scientifiques, etc.

## 1.3 Formalismes de représentation des données semi-structurées

### 1.3.1 Le modèle OEM (*Object Exchange Model*)

Le modèle OEM a été le premier modèle le plus utilisé pour représenter les données semi-structurées. Il est issu du projet TSIMMIS de l'Université de Stanford [Papakonstantinou et al., 1995]. Dans ce modèle, les données sont auto-descriptives et modélisées sous forme de graphes étiquetés et orientés. Chaque donnée du modèle OEM peut être vue comme une collection d'objets que l'on retrouve sous forme de nœuds. Chaque objet a un OID unique (*Object Identifier*) et peut être atomique ou complexe. Les objets atomiques contiennent une valeur qui appartient à un des types de base atomiques tels que les types entier, réel, chaîne, gif, son, etc. Les autres types d'objets sont complexes, c'est-à-dire formés à partir des types de base. Leurs valeurs sont un ensemble de couples (étiquette, oid), où les étiquettes sont des chaînes de caractères identifiant un nom d'attribut et les oids sont des identifiants d'objets. Le graphe comporte un nœud particulier qui est appelé racine d'une base OEM.

La figure 1.1 donne un exemple de donnée OEM où :

- &1, &2, &3, etc. sont des identifiants d'objets (oids) ;
- La racine &1, désignée dans le schéma par l'étiquette 'MaC0', regroupe trois objets de type complexe ; les deux premiers sont étiquetés "employé" et le troisième "département" ;
- &5 est un exemple d'un objet simple étiqueté "nom", cet objet est de type chaîne de caractères et de valeur "Jean".

### 1.3.2 La modèle XML

XML [Bray et al., 1998] (*eXtensible Markup Language*) est un langage à balises étendu, appelé aussi langage à balises extensible. C'est une simplification du langage SGML [Goldfarb, 1990] mais il peut être vu comme un langage HTML amélioré par la possibilité de définir de nouvelles balises. Il s'agit effectivement d'un langage permettant de mettre en forme le contenu des documents grâce à des balises (*tags*).

Contrairement à HTML, qui doit être considéré comme un langage défini et figé (avec un nombre de balises limité), XML peut être considéré comme un métalangage permettant de définir d'autres langages, au moyen de nouvelles balises nécessaires à la description d'un texte ou d'une donnée semi-stucturée. La force de XML réside dans sa capacité à pouvoir décrire des données de n'importe quel domaine grâce à son extensibilité. Il permet de structurer et de poser le vocabulaire d'un domaine, et ainsi définir la syntaxe des données qui seront utilisées. Les balises XML décrivent le contenu plutôt que la présentation (contrairement à HTML). Ainsi, XML permet de séparer le contenu de la présentation. Ceci permet par exemple d'afficher un même

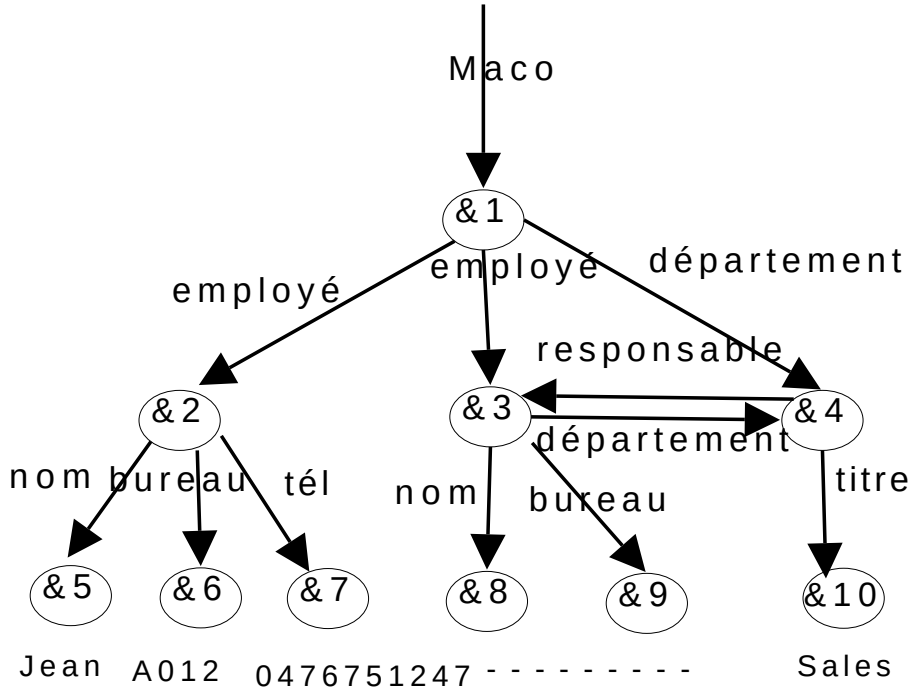


FIG. 1.1 – Exemple d'un graphe OEM

document sur des applications ou des périphériques différents sans pour autant nécessiter la création d'autant de versions du document que l'on veut de représentations.

Les documents XML utilisés dans notre domaine d'application qui est le domaine médical et plus précisément "Hôpital", définissent clairement et en détail tous les notions (données) nécessaires dans la structure "Hôpital".

Pour qu'on puisse interroger des données XML, on est amené à étudier en premier lieu leur structuration.

### Structure d'un doncument XML

Un document XML est structuré en trois parties :

**La première partie**, appelée prologue, permet d'indiquer la version de la norme XML utilisée pour créer le document ainsi que le jeu de caractères (*encoding*) utilisé dans le document. La version est un attribut obligatoire du prologue tandis que le jeu de caractères est un attribut facultatif. Un exemple de prologue est donné dans la ligne suivante :

```
<? xml version="1.0" encoding="ISO-8859-1"?>
```

Ce prologue spécifie qu'il s'agit de la version XML standard "1.0" et le jeu de caractères que l'on va utiliser est le jeu "ISO-8859-1", qui prend en compte les accents français. Par exemple dans notre domaine les balises : <rue> Jeanne d'Arc</rue>. Voir annexe A pour plus de détails. Le prologue se poursuit avec des informations facultatives sur des instructions de traitement à destination d'applications particulières. Leur syntaxe est la suivante :

```
<?instruction de traitement?>
```

**La seconde partie** est une déclaration de type de document (à l'aide d'un fichier annexe appelé DTD - *Document Type Definition*. Cette DTD impose une structure type prédéfinie (une forme de grammaire) au document XML dans lequel elle est déclarée. Il existe deux types de déclarations d'une DTD :

1. **Déclaration intérieure à une DTD :** Si la déclaration d'une DTD est faite à lintérieur de la DTD, elle doit être déclarée dans une définition de DOCTYPE, avec la syntaxe suivante :

```
<!DOCTYPE nom-de-la-racine [déclarations-des-éléments]>
```

2. **Déclaration extérieure à une DTD :** Si la déclaration d'un DTD est définie dans un fichier extérieur, elle doit être déclarée dans une définition de DOCTYPE, avec la syntaxe suivante :

```
<!DOCTYPE  nom-de-la-racine SYSTEM "nom-de-ficher">
```

**La dernière composante** d'un document XML est l'arbre des éléments. L'arbre des éléments, qui constitue le véritable contenu du document XML, est constitué d'une hiérarchie de balises comportant éventuellement des attributs. Un attribut est un couple <nom_attribut, valeur> écrit sous la forme : nom_attribut= "valeur". Ainsi, une balise affectée d'un attribut aura la syntaxe suivante : <balise nom_attribut="valeur">

En sachant qu'une donnée peut éventuellement être un ensemble d'éléments XML, toute donnée est encapsulée entre une balise ouvrante <balise> et une balise fermante </balise>. Ainsi, un élément vide est uniquement constitué d'une balise spécifique dont la syntaxe est la suivante : <balise/>. D'autre part, il est interdit en XML de faire chevaucher des balises, c'est-à-dire d'avoir une succession de balises du type :

```
<balise1><balise2></balise1></balise2>
```

Par contre, il est possible entre les balises (mais pas à l'intérieur d'une balise) d'ajouter des espaces, des tabulations et des retours chariots, ce qui est très utile pour définir une indentation des balises. Voici un exemple très simple de document XML :

```
<annuaire>
   <personne class = "etudiant">
<nom>Pillou</nom>
<prénom>Jean-Francois</prénom>
<téléphone>555-123456</téléphone>
<email>webmaster@commentcamarche.net</email>
   </personne>
</annuaire>
```

**Documents bien formés et documents valides :** Un document est dit "bien formé" lorsqu'il obéit aux règles syntaxiques XML. Il est écrit correctement et il ne provoquera pas d'erreurs lorsqu'il sera utilisé. Un document est "valide", en plus d'être "bien formé",lorsqu'il est conforme à une structure type définie explicitement dans une DTD (Document Type Definition).

## 1.4 Méta-données pour les données semi-structurées

La souplesse d'utilisation des données XML a pour contrepartie les inconvénients suivants :
- le stockage des données est inefficace.
- les requêtes sont complexes à formuler.
- les requêtes sont complexes à évaluer de façon optimale.

Afin d'apporter des solutions à ces inconvénients, des recherches ont été effectuées pour décrire et exploiter la régularité que possèdent souvent les données semi-structurées (données XML). Deux approches ont vu le jour :
- un schéma déduit : calculé automatiquement a posteriori à partir des données. On peut citer les guides de données [Goldman & Widom, 1997], les T-indexes [Milo & Suciu, 1999] ;
- un formalisme de schéma flexible : défini a priori. De telles formulations ont fait l'objet d'études et de standardisation, notamment DTD [Bosak et al., 1998] et XML-Schema [Thompson et al., 2001].

### 1.4.1 Première direction de recherche (Les guides de données)

Selon [Sebi, 2007], Les premiers guides de données ont été implémentés dans le cadre du SGBD de données semi-structurées Lore (*Lightweight Object Repository*) [McHugh et al., 1997] à l'université de Stanford. Le modèle de données utilisé dans Lore est le modèle OEM (*Object Exchange Model*) [Papakonstantinou et al., 1995]. Le langage utilisé pour interroger une base de données Lore est le langage Lorel [Abiteboul et al., 1996], qui est inspiré de OQL. Lore stocke les guides de données comme des objet OEM. Ils sont générés d'une manière dynamique et ils sont conformes aux données.

Un guide de données est un résumé structurel, à la fois précis et concis, d'un graphe de données G. Il est précis dans le sens où tous les chemins apparaissant dans G sont contenus dans le guide, et concis dans le sens où chaque chemin de G apparaît au plus une fois dans le guide. Le guide de données peut être utilisé comme un index de chemins, mais aussi comme un auxiliaire précieux pour la formulation de requêtes, en fournissant à l'utilisateur, par l'intermédiaire d'une interface graphique, une vue synthétique de l'ensemble des chemins d'un graphe de données.

**Définition** *Guide de données* : Un guide de données G pour une base OEM est un objet OEM g tel que chaque chemin de G, compris comme séquence d'étiquettes, apparaît une fois et une seule comme un chemin de g, et chaque chemin de g est un chemin de G [Nestorov et al., 1997], [Hopcroft et al., 2001].
La figure 3.2, extraite de [Papakonstantinou et al., 1995], présente une base de données OEM très petite, qui représente une partie d'une base de données d'un guide de restauration. Nous pouvons voir que la structure de la base de données est irrégulière, puisque le restaurant " Darbar", avec deux entrées, n'inclut aucune information sur le numéro de téléphone. En conclusion, on note que les bases de données OEM n'ont pas besoin d'avoir une structure arborescente. Dans l'exemple, l'objet "Smith" est propriétaire d'un restaurant et directeur d'un autre.

La figure 1.3 montre un guide de données pour la base de données OEM de la figure 3.2 ; notons que un guide de données ne contient aucune valeur atomique. Puisqu'un guide de données est prévu pour refléter la structure d'une base de données, les valeurs atomiques sont inutiles.

La définition d'un guide de données ne suffit pas pour assurer l'unicité d'un guide de données pour une base, ce que montre la figure 1.4 en donnant deux exemples de guides de données (b)

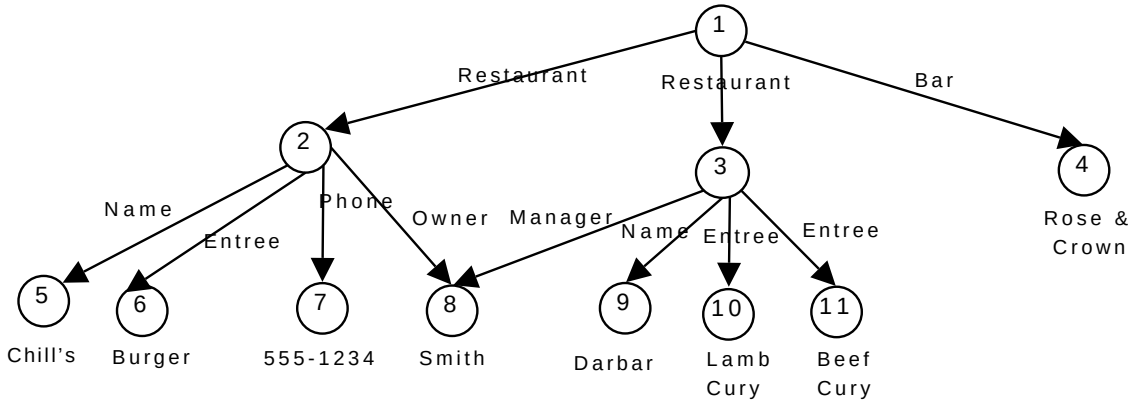


FIG. 1.2 – Exemple de base de données OEM

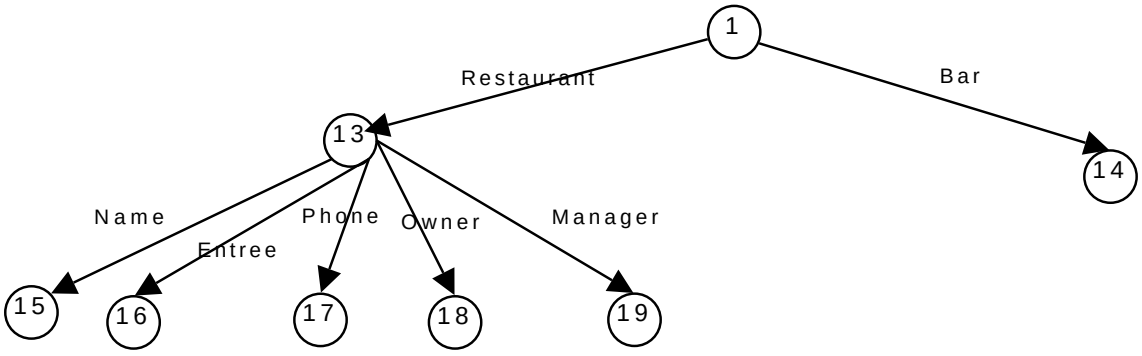


FIG. 1.3 – un guide de données pour la base 3.2

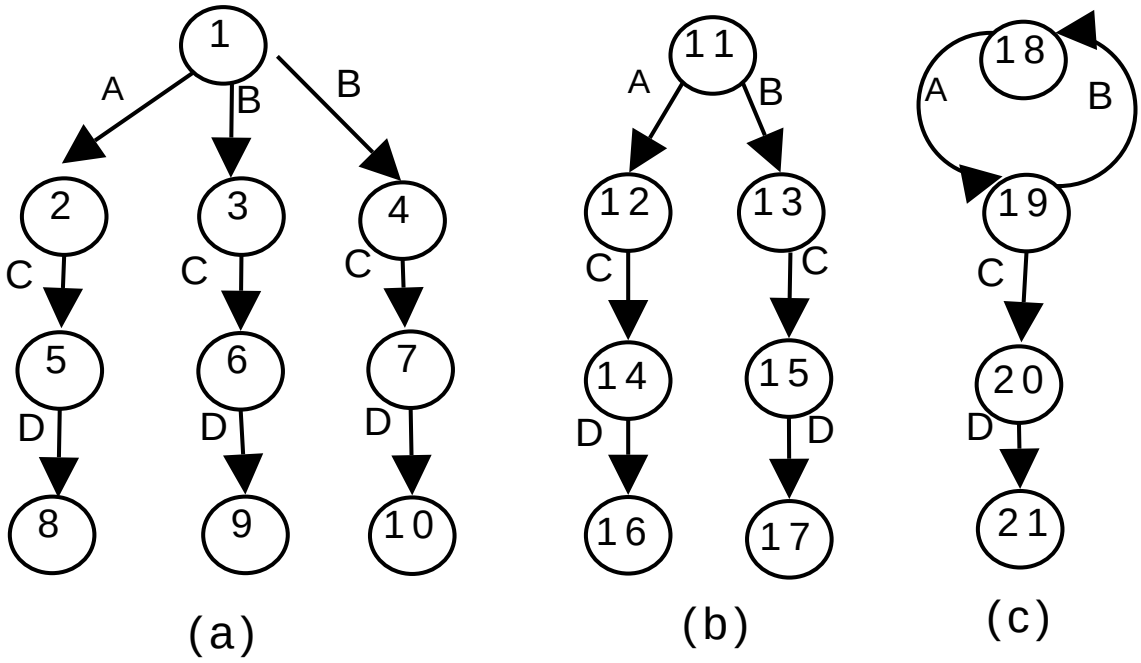


FIG. 1.4 – Une base de données orginale et deux guides de données

et (c) pour la base de données (a). En utilisant la théorie des automates finis, on montre que le problème de la construction d'un guide de données se ramène à celui de la conversion d'un automate fini non déterministe en un automate fini déterministe. Ce parallèle permet d'assurer l'existence de plusieurs guides de données pour une même base. Il est donc nécessaire de choisir un guide de données qui possède des propriétés intéressantes, notamment en ce qui concerne les mises à jour et les possibilités d'utilisation comme index des chemins.

L'étude [Goldman & Widom, 1997] montre que le guide minimal (le guide C dans l'exemple) n'est pas le plus intéressant car sa maintenance incrémentale est en général coûteuse, parce qu'il est nécessaire de revoir la construction de tout le guide de données pour chaque mise à jour.

Pour pallier ces inconvénients, la notion de guide de données *robuste* est définie. Intuitivement, l'idée est de construire un guide g tel que pour tout ensemble e de chemins de g qui aboutissent au même nœud de g, e conduit au même ensemble d'objets dans le graphe de données.

En résumé, un guide de donnée *robuste* constitue un outil particulièrement intéressant pour l'exploitation des bases de données semi-structurées. D'une part, il peut être utilisé comme un résumé structurel des données, afin de faciliter la formulation des requêtes, et d'autre part il permet l'évaluation efficace d'expressions de chemins simples depuis la racine du graphe.

En revanche, Il présente deux inconvénients : le premier est qu'il n'est pas adapté à l'évaluation d'expressions de chemins généralisées. Le deuxième inconvénient est la taille exponentielle des guides de données par rapport à la taille du graphe de données dans le pire des cas.

Ces inconvénients nous ont conduits à suivre la deuxième direction de recherche (décrite par la suite) et qui se base sur la définition d'un schéma flexible.

### 1.4.2 Deuxième direction de recherche (DTD, Schéma XML)

#### DTD (*Document Type Definition*)

Les documents valides obéissent à une structure type prédéfinie nommée DTD (*Document Type Definition*). Un document valide sera une instance de la structure décrite par une DTD. Produire des documents valides présente un double intérêt. D'abord, l'auteur d'un document XML évite de réinventer des structures complexes, déjà disponibles, validées par de nombreux utilisateurs : une DTD est réutilisable. Le second avantage est que les documents peuvent être validés grâce à un parseur validant. Un parseur est un programme qui vérifie la syntaxe XML du document, le respect de la structure définie par la DTD et le fait que toutes les références à des entités peuvent être résolues.

En revanche, ce format de description, hérité de SGML, présente de nombreux handicaps :

1. les DTDs ne sont pas au format XML. Cela signifie que pour analyser (*parse*) un tel document XML, il est nécessaire d'utiliser un outil différent de celui utilisé pour l'édition d'un document XML.
2. les DTDs n'offrent pas de support pour les "espaces de noms". En pratique, cela implique qu'il n'est pas possible, dans un document XML défini par une DTD, d'importer des définitions de balises définies par ailleurs.
3. le "typage" des données est extrêmement limité.

Conçu pour pallier les handicaps des DTDs cités ci-dessus, le Schéma XML propose de nouvelles fonctionnalités en plus de celles fournies par les DTD. Ainsi, sont introduits :

- Le typage des données. Il permet la gestion de booléens, d'entiers, d'intervalles de temps, Il est possible de créer de nouveaux types à partir de types existants.
- La notion d'héritage. Les éléments peuvent hériter du contenu et des attributs d'un autre élément. C'est sans aucun doute l'innovation la plus intéressante de Schéma XML.
- Le support des espaces de noms.
- Les indicateurs d'occurrence des éléments. Ils peuvent être un nombre non négatif quelconque (rappel : dans une DTD, on était limité à 0, 1 ou un nombre infini d'occurrences pour un élément).
- La possibilité de définition modulaire des schémas. Les schémas XML sont très facilement concevables par modules.

#### Les Schémas XML

Le but d'un schéma est de définir une classe de documents XML. Un schéma permet de décrire les autorisations d'imbrication et l'ordre d'apparition des éléments et de leurs attributs, tout comme une DTD. Mais il permet aussi d'aller au-delà. Ainsi, un premier point intéressant est qu'un Schéma XML est lui-même un document XML, ce qui permet à un tel schéma XML d'être manipulé de la même manière que n'importe quel autre document XML, en particulier par une feuille de style XSL. Par exemple, il est possible d'automatiser la création d'une documentation à partir d'un schéma, sur la base des commentaires et explications qui s'y trouvent. Un deuxième point est qu'un schéma permet aux programmeurs des applications échangeant de données sur le Web de réaliser des codes robustes capables d'éviter ou de détecter les erreurs de codage, en déclarant des structures de données " fortement typées ".

Dans notre cas d'étude, les données XML sont utilisées dans un projet d'interrogation de base de données afin de répondre aux requêtes des utilisateurs dans un temps optimal. Par la suite on fait une description des langages d'interrogation pour XML.

## 1.5 Langages d'interrogation pour XML

L'interrogation de documents semi-structurés de type XML nécessite la définition d'un nouveau langage de requêtes. Dans cette section, nous examinons tout d'abord les fonctionnalités souhaitables pour un langage de requêtes pour XML, puis nous présentons quelques langages typiques avant d'aborder plus loin le standard en émergence XQuery.

### 1.5.1 Objectifs d'un langage de requêtes pour XML

Les bases de données XML se situent à la croisée des chemins des systèmes documentaires et des bases de données [Gardarin, 2002] ; un langage de requêtes doit donc intégrer les fonctionnalités des langages de requêtes de ces deux types de systèmes. Ainsi, ce qui est attendu d'un tel langage de requête est :

- la possibilité d'effectuer toutes les opérations propres aux bases de données relationnelles ; c'est pourquoi les opérateurs habituels dans les bases de données (projection, sélection, jointure, etc.) doivent être définis ;
- la possibilité de naviguer dans les données : le langage doit intégrer des notions de chemins ;
- la recherche par mots-clés : l'intégration des fonctions des systèmes documentaires nécessite la prise en compte de requêtes par mot-clés ;
- la construction du résultat : on doit pouvoir spécifier la façon dont le résultat devra être présenté ;
- la possibilité d'interroger simultanément le schéma et les données : on doit pouvoir combiner dans la même requête des interrogations sur les éléments du schéma et des interrogations sur les données.

Plusieurs langages, répondant plus ou moins à ces besoins, ont été proposés pour permettre l'interrogation de données semi-structurées. En 1994, [Christophides et al., 1994] ont proposé les langages SGMLQL, HyOQL et LOREL : ces langages sont des extensions du langage OQL qui permettent de traverser des graphes généralisés et d'effectuer des recherches dans les contenus des documents XML. En 1995, le langage fonctionnel SGMLQL, qui permet de poser des requêtes sur des documents SGML, a été proposé par [Maitre et al., 1997]. HyOQL [Gardarin & Yoon, 1996a] propose des algorithmes pour traverser des graphes ; LOREL/OEM-QL [Abiteboul et al., 1998] est un langage qui s'appuie sur le modèle de données OEM et qui suppose une connaissance préalable des expressions de chemin Lorel. LOREL/OEM-QL permet la construction de graphes généralisés sur un modèle OEM.

Parmi les autres langages qui ont été proposés, on signalera le langage XML-QL [Deutsch et al., 1999], le langage Quilt proposé par [Chamberlin et al., 2001], et le langage YATL proposé par [Cluet et al., 1998] à l'INRIA. On remarque que, malgré la différence entre ces langages, ils ont un point commun, inspiré par SQL, qui est de décomposer une requête en trois parties [Aguilera, 2001] :

- un *pattern* (*motif*), qui peut être vu comme la définition d'une relation R. Il permet d'associer des variables aux portions de documents concernées par la requête.
- un constructeur, qui définit une structure complexe pour chaque tuple sélectionné de la relation R.
- un filtre, qui sélectionne certains tuples de R en utilisant des prédicats.

Dans la suite, on présentera le langage XQuery, qui a été fortement inspiré par ces travaux. On commence par analyser comment intégrer un document XML dans un SGBD. Cette analyse nous servira par la suite dans notre recherche, à savoir comment on peut transformer un document XML dans un schéma relationnel afin de stocker les données XML dans des tables relationnelles

pour pouvoir les interroger en utilisant le langage de requête SQL. Après cette analyse on considèrera les extensions apportées à SQL pour traiter les documents, puis on présentera XPath, brique de base de nombreuses autres recommandations du W3C, pour finir par XQuery.

### 1.5.2 Importation de documents XML dans un SGBDR

Dans cette section nous sommes fortement inspirés de l'état de l'art étudié dans [TOPER, 2007].

XML est un modèle qui représente des données semi-structurées, cependant SQL interroge des données structurées. Ces deux types de données sont différents mais sont-ils incompatibles ? Si la réponse est oui, SQL ne peut pas interroger des documents XML ; dans le cas contraire, la question doit être approfondie.

On peut modéliser une donnée structurée sous une forme semi-structurée, car les données semi-structurées sont des données dotées de propriétés moins fortes que les données structurées. Dès lors, XML devient un format pivot. C'est pourquoi les SGBDR utilisent XML pour faire migrer des données d'une base de données vers une autre. XML (associé à XML Schéma) permet de détecter et de traiter les débordements potentiels liés aux types. Par exemple, lors d'un transfert on peut détecter qu'un champ *Integer* du SGBDR source a une valeur plus élevée que celle permise par le SGBDR destinataire.

Pour analyser la compatibilité de ces deux formats (XML et les tables dans un SGBDR), il est nécessaire d'étudier comment importer un document XML dans un SGBDR et de régénérer ce document à partir du SGBDR. Ce *roundtripping* [1] permet de montrer qu'aucune information n'est perdue ou altérée, à condition que le document XML soit représentatif d'un modèle de structure. L'importation d'un document XML dans un SBGDR est appelée *shredding* : Le document XML est découpé pour être intégré dans différentes tables [Florescu & Kossmann, 1999].

Différents algorithmes existants permettent d'importer un document XML dans un SGBDR, en concevant le document comme un arbre. Cependant, l'exploitation du document sous forme relationnelle n'est pas optimisée : la performance (espace et temps d'exécution) n'est pas optimale et surtout l'interrogation du document devient plus complexe. Ainsi, en utilisant les *nested sets*, il est trivial de modéliser n'importe quel arbre XML, mais l'interrogation est compliquée.

Par exemple, considérons une partie du document XML utilisé dans notre expérimentation :

```
<bib>
  <book year="1994">
    <title>TCP/IP Illustrated</title>
    <author name="W.Stevens" ></auteur>
    <publisher>Addison-Wesley</publisher>
    <price>65.95</price>
  <book  year="2000">
     <title>Data on the Web</title>
     <author name="Abiteboul"</author>
     <publisher>Morgan Kaufmann </publisher>
     <price>39.95</price>
</bib>
```

Il peut être transformé dans le schéma relationnel suivant :

[1] possibilité de convertir des données dans une structure différente et de les convertir de nouveau dans la structure originale, sans perte d'information

```
books (idbook, author, price); \newline
book (id, author, publisher, price);\newline
author (id, name);
```

La requête Xpath suivante renvoie le titre des livres dont l'auteur est 'Abiteboul' :

```
//author[@name='Abiteboul']
```

cette requête s'exprime sous la forme SQL suivante :

```
SELECT *
FROM book, books, author
WHERE books.idbook='4' AND book.id=books.idbook AND book.author=author.id AND
                                                    author.name=' Abiteboul '
```

Enfin, d'après [Florescu & Kossmann, 1999] et [Melton & Buxton, 2006], le document XML régénéré est différent du document initial, au moins pour les informations de peu de valeur comme les espaces. Certains documents complexes ou les documents narratifs se prêtent mal au *shredding*.

### 1.5.3 SQL/XML : extension de SQL

SQL/XML est une extension de la norme SQL qui fournit le support à l'usage de XML dans le contexte d'un système de base de données relationnelles. Puisque SQL est le langage standard pour l'accès et la gestion de données stockées dans les bases de données relationnelles, il est normal que les entreprises et les utilisateurs dans le monde entier aient la possibilité d'intégrer leurs données XML dans les données existantes qui leur sont apparentées, par l'utilisation des équipements de SQL. SQL/XML permet de stocker les documents XML dans une base de données SQL, d'interroger ces documents en utilisant XPath et XQuery, et d'*éditer* les données relationnelles existantes sous forme de documents XML. Considérons le document XML suivant :

```
<?xml version="1.0" encoding="ISO-8859-1" ?>
<Guide region="Normandie" version="2.0">
   <Restaurant type="français" categorie="***">
      <Nom>Le Grand Hôtel</Nom>
      <Adresse>
           <Rue>Promenade M.Proust</Rue>
           <Ville>Cabourg</Ville>
     </Adresse>
     <Prix>300</Prix>
  </Restaurant>
  <Restaurant type="français" categorie="**">
     <Nom>L'Absinthe </Nom>
     <Adresse>
          <No>Promenade M.Proust</No>
          <Rue>qui Quarantaine</Rue>
          <Ville>Honfleur</Ville>
     </Adresse>
     <Prix>250</Prix>
     <Spécialité>Fruits de Mer</ Spécialité>
   </Restaurant>
</Guide>
```

SQL/XML s'appuie sur les expressions de navigation XPath. Les expressions XPath absolues sont utilisées à la place des tables : elles permettent de désigner par des variables les arbres de la forêt sélectionnée par l'expression de chemin en descendant dans les documents depuis la racine. Le critère de sélection désigné par la clause WHERE s'applique sur des éléments de ces arbres. Les sous-arbres sélectionnés sont simplement agrégés par un constructeur *séquence*, ce qui ne permet pas de reconstruction d'arbres. La requête suivante, par exemple :
" Lister le nom des restaurants de Grenoble", s'écrit en SQL/XML :

```
SELECT  $R/Nom
FROM      /Guide/Restaurant $R
WHERE   $R/Ville="Grenoble
```

SQL/XML permet d'interroger un document XML en utilisant XQuery et XPath. Le programmeur d'application doit connaître ces deux langages. Ce langage permet aux SGBDR de devenir des entrepôts de documents XML et aux applications de bénéficier de toute la solidité des SGBDR, dont les fonctions de gestion. XML a été conçu pour des données semi-structurées et SQL pour des données structurées. Ces différences conceptuelles expliquent les difficultés à lier les deux d'un point de vue théorique. Mais les extensions de SQL telles que SQL/XML intègrent d'autres langages afin d'assurer une interopérabilité entre ces deux formats et un stockage simple des documents XML.
En conclusion, il est possible d'une manière générale d'interroger un document XML via un SGBDR, et donc en utilisant SQL. Mais soit le code SQL est complexe, soit il utilise plusieurs langages. D'autre part, les bases de données relationnelles constituent une technologie de stockage et d'interrogation pérenne. Une solution à envisager est la traduction de requêtes XPath ou XQuery en SQL [Krishnamurthy et al., 2003].

### 1.5.4 XPath 1.0

#### Concepts principaux

D'après la recommandation du W3C, le principal objectif de XPath est de définir la manière d'accéder à des parties d'un document XML. Afin de satisfaire ce premier objectif, la recommandation du W3C fournit également les moyens pour traiter des chaînes de caractères, des nombres et des booléens. En plus de l'adressage des données, XPath est également conçu pour répondre aux besoins d'adressage de XSLT et XPointer.

Un autre objectif de XPath est de permettre la sélection d'un ensemble de nœuds en utilisant de fonctions de sélection du langage d'interrogation d'un document XML et la navigation dans ce document. XPath représente les documents XML comme un arbre de nœuds. Il y a plusieurs types de nœuds, parmi lesquels les nœuds d'éléments, les nœuds d'attributs et les nœuds de texte.

Une expression XPath caractéristique est un chemin de localisation constitué par une suite d'éléments ou d'attributs, séparés par une barre de fraction (" / "), ressemblant au chemin dans un système de fichiers. XPath fournit des fonctions intégrées, permet d'utiliser des variables, de définir des filtres et de spécifier des axes. Les chemins de localisations sont divisés en étapes qui ont chacune trois composants :

- un axe : indique le type de l'information qui sera sélectionnée, relativement au nœud courant ou depuis la racine.

[1] W3c, Recommendation Xpath 1.0, http ://www.w3.org/TR/xpath

- un nœud de test : un nœud de test définit les éléments ou attributs à désigner.
- des prédicats : ce sont des expressions plus complexes ; ils sont utilisés pour filtrer ou exclure certains nœuds.

Voici quelques exemples des expressions XPath avec prédicats :
CINEMA [count (SEANCE) > 1] : renvoie tous les cinémas avec au moins 2 séances ;
FILM [count (ACTEUR) = 0] : renvoie tous les films sans acteurs ;
FILM [not (ACTEUR)] : renvoie tous les films sans acteurs.

#### XPath comme langage d'interrogation et ses limites

D'après [TOPER, 2007] il existe deux manières d'interroger un document XML en utilisant XPath : grâce aux chemins, en utilisant les méta-données de XML, ou grâce aux prédicats qui permettent de filtrer les résultats. La recommandation de Xpath a été approuvée en 1999. Différentes implantations ont été écrites et utilisées. Toutefois, Les versions de XPath ont des limites, Tout d'abord, ils ne permettent pas la combinaison d'un ensemble de sources de données. Par exemple, joindre deux documents XML est impossible avec XPath, Or cette fonctionnalité est requise dans beaucoup de cas d'utilisation. Par ailleurs, XPath fonctionne mal pour de gros documents, parce qu'un moteur d'interrogation XPath doit analyser un document en entier afin d'être en mesure d'apporter une réponse à toutes les requêtes possibles.

En conclusion, XPath a été conçu pour être utilisé en conjonction avec d'autres langages. Son objectif est de sélectionner une partie d'un document, ce qui ne représente qu'une partie des cas d'utilisation de l'interrogation de documents XML.

Dans notre cas d'étude on a fait appel à la version 2.0 de XPath afin de pouvoir récupérer tous les nœuds feuilles de l'arbre DOM d'un document XML dans notre base source de données.

### 1.5.5 XQuery

XQuery est une amélioration de XPath ; c'est un langage qui a été conçu pour poser des requêtes sur un ou plusieurs documents XML. En utilisant XQuery, on peut rechercher des documents entiers, des sous-sections des documents, y compris des valeurs trouvées sur différents nœuds d'un document. On peut également l'utiliser pour transformer des valeurs retournées par des requêtes posées sur un document. XQuery considère un document XML comme une collection d'éléments, de texte, et de noeuds d'attribut.

Nous définissons ci-dessous quelques concepts de base utilisés dans XQuery.

**Prédicats** Quand on fournit le chemin d'une requête XQuery, la réponse est constituée d'un ensemble de résultats. On peut filtrer cet ensemble de résultats en utilisant des prédicats. Les prédicats sont toujours entre parenthèses ([ ]) et il y a deux types de prédicats : numérique et booléen.

**Prédicats Numériques** : Un prédicat numérique permet de choisir un nœud basé sur sa position relativement à un autre nœud dans le document (c'est-à-dire, basé sur son contexte). Par exemple, l'expression XQuery suivante : /nœud1/nœud2/nœud3 retourne la première valeur associée à <nœud3> dans le document. Pour renvoyer par exemple le deuxième élément de <Nœud3>, on utilise le prédicat /Nœud1/Nœud2/Nœud3[2].

**Prédicats Booléens :** Un prédicat booléen filtre un résultat de requête de telle manière qu'on garde seulement les éléments du résultat qui satisfont l'expression évaluée (l'expression est vraie pour ces éléments). Si l'on souhaite choisir un nœud seulement si son nœud de type texte est

égal à une certaine valeur, on utilisera le prédicat booléen suivant : /Nœud1/Nœud2[Nœud3="Nœud3 texte 3"]

***Contexte*** : la signification d'une expression XQuery peut changer selon le contexte courant. Dans des expressions XQuery, le contexte n'est important que si l'on veut utiliser les chemins relatifs ou si les documents emploient des namespaces.

***Chemins (paths) relatifs*** : n'importe quel chemin qui ne commence pas par un slash (/) est relatif à l'endroit courant dans un document. L'endroit courant dans un document est déterminé par le contexte.

***Wildcards*** : XQuery permet d'employer des *wildcards* quand les éléments de document sont inconnus. Par exemple : /Nœud0/*/Nœud6 - choisit tous les nœuds Nœud6 qui sont des nœuds au troisième niveau de profondeur dans le document et dont le chemin démarre par Nœud0. D'autres exemples de wildcards sont :

- Choisir tous les nœuds dans le document : //*
- Choisir tous les nœuds Nœud6 qui ont trois ancêtres : /*/*/*/Nœud6
- Choisir tous les nœuds immédiatement sous Nœud5 : /Nœud0/Nœud5/*
- Choisir tous les attributs de Noeud5 : /Nœud0/Nœud5/@*

***Fonction de navigation*** : XQuery fournit plusieurs fonctions qui peuvent être employées pour la navigation globale à un document ou à une collection spécifique de documents.

***Expression FLWOR*** : XQuery offre des possibilités d'itération et de transformation par les expressions "FLWOR". FLWOR est un acronyme qui représente les cinq clauses principales dans une expression de FLWOR : For, Let, Where, Order by and Return. En utilisant les expressions "FLWOR", on peut faire des itérations sur des séquences, utiliser les variables, le filtre et le groupe. On peut aussi utiliser FLWOR pour exécuter des jointures de différentes sources de données. Par exemple, en supposant que l'on a des documents XML de la forme :

```
<produit>
   <nom>Jus</nom>
   <prix>0.83</prix>
</produit>
```

On peut ordonner tous ces documents XML qui sont dans un même container par prix en utilisant l'expression FLWOR suivante :

```
for $i in collection("myContainer.dbxml")/produit
order by $i/prix descending
return $i
```

Avec la prolifération de documents XML est apparue la nécessité des requêtes sur XML. SQL n'est possible que sur des modèles de données relationnelles, c'est pourquoi un nouveau langage de requêtes avec des capacités similaires mais sur des types de données différents était nécessaire. Après des années de travail dans le cadre du W3C, les spécifications de XQuery ont permis de répondre à ces besoins.
Après avoir présenté les différents formalismes de données semi-structurées et leurs langages d'interrogation, nous présentons par la suite les systèmes de gestion de bases de données pour les données XML.

## 1.6 SGBD XML (*XML Data Base*)

La croissance permanente des données semi-structurées, dont le standard est le format XML, a conduit par ailleurs à définir des systèmes de gestion de bases de données (SGBD) adaptés

pour ce type de données.

### 1.6.1 Classification architecturale des systèmes de gestion de bases de données XML

Les systèmes de gestion de bases de données ont trois architectures principales.

1. **L'approche Relationnelle** :
   Comme les systèmes de gestion de bases de données relationnelles sont très répandus, ils sont souvent utilisés avec XML. Dans l'approche relationnelle, XML est mis en correspondance (*is mapped*) avec un schéma relationnel. Par exemple, considérons ce modèle pour un document XML avec une structure simple :

   ```
   A <book>is made up of <chapters>
   A <chapter>is made up of<sections>
   A <section>is made up of<paragraphs>
   A <paragraph>is made up of text.
   ```

   Il est facile de mettre ce document, strictement hiérarchique, dans un système de gestion de bases de données relationnelles, en utilisant des clés étrangères :

   ```
   Create Book
    {
      book_isbn       char(256)   key
      book_name       char(256)
      book_chapter_no int

   Create Chapter
    {
       chapter_no   int
       chapter_section_no  int
    }
   Create Section
   {
     section_no   int
     section_para_no  int
   }
   Create Para
     Para_no  int
     Para-text char varying (32768)
   }
   ```

   "Book" peut être créé ou recréé en utilisant des opérations de jointure. Ainsi, pour créer un "Book" entier, on peut utiliser la jointure suivante :

   ```
   SELECT (para_text)
   From Book, Chapter, Section, Para
   WHERE
     ((book_isbn=?) AND
     (book_chapter_no=chapter_no)AND
     (chapter_section_no=section_no)AND
     (section_para_no=para_no))
   ```

```
ORDER BY chapter_no, section-no,para_no ascending
```

On note que quand le document devient plus complexe ou quand le modèle de données contient beaucoup de niveaux de données hiérarchiques, la jointure devient beaucoup plus grande et exige beaucoup de clefs étrangères. Les traitements des instructions, les définitions de type de documents (DTDs) et les commentaires ne sont pas compris dans cette approche.

2. **L'approche basée sur le texte (*Text-Based Approach*)** Certains systèmes de gestion de données XML (*SGBD-XML*) stockent les données XML comme des chaînes de caractères. Pour accélérer les recherches dans ces chaînes, le *SGBD-XML* indexe le texte ; cette indexation organise les données d'une manière efficace pour faciliter leur récupération et permettre d'accéder facilement à certaines parties des documents.
3. **L'approche orienté-objet** (*Object-Based Approach*) Dans cette approche, le document XML réside dans la base de données sous un format d'objet. Chaque article devient un objet. Par exemple, chaque élément devient un objet et chaque attribut dans un élément devient un attribut dans l'objet. La relation entre les éléments, leur position dans la hiérarchie, leur ordre, etc. sont maintenus.

### 1.6.2 Evaluation des solutions

Les solutions relationnelles, texte et orienté-objet ont des avantages et des limites que le programmeur doit exploiter ou adapter aux situations. Dans la suite on présente ces avantages et ces limites pour chacune des approches :

**L'Approche Relationnelle** : A première vue, l'usage des SGBDs relationnels pour le stockage de données XML est très attrayant parce qu'ils offrent toutes les caractéristiques des systèmes relationnels. Mais cette solution s'avère peu adaptée, car la distance entre la structure relationnelle (tables, relations, lignes et colonnes) et la structure hiérarchique des données XML rend complexe la manipulation des données XML avec les opérateurs relationnels. En fait, plus la puissance de XML et sa flexibilité sont exploitées, plus il est difficile de stocker des données XML dans une base de données relationnelle.

Considérons des documents XML plus réalistes, mais toujours avec une structure simple :

```
A <Book> is made up of <Chapters>
A <Chapter> is made up of <Section> or <Paragraphs>
A <Section>is made up of <Paragraphs>
```

Il est très difficile de mettre en application ce modèle dans un système relationnel. L'application peut inclure des balises <Section> même quand elles ne sont pas nécessaires, mais dans ce cas, le document stocké ne reflète pas la structure désirée. Le document ne peut pas être créé en utilisant une jointure simple.

**L'approche basée sur le texte** : En utilisant cette approche les opérations de stockage et de récupération sont très rapides, parce que la manipulation de chaînes de caractère exige seulement la copie de données dans ou depuis la base de données. Les requêtes sont en général rapides, parce que le texte peut être indexé en utilisant les technologies standard de recherche dans les textes. Cependant, la maintenance est difficile. Changer la valeur d'un attribut, par exemple, exige de copier la chaîne de caractères entière dans la mémoire, de rechercher l'occurrence appropriée et de changer les données.

Si les nouvelles données sont différentes, une programmation supplémentaire est nécessaire pour manipuler les données plus longues. De plus, en renvoyant le texte XML au système de gestion de bases de données, la chaîne entière doit remplacer l'ancienne, et le document doit être réindexé. Quand les données sont majoritairement des textes et nécessitent peu de mises à jour, les solutions ci-dessus peuvent être appropriées.

**L'approche orienté-objet** : Les solutions orienté-objet offrent de bonnes caractéristiques de traitement dans les situations (*text-related*) et (*non text-related*), parce qu'elles accèdent directement à n'importe quelle partie des ces éléments, attributs ou données.

En utilisant cette approche, les opérations de lecture de documents et les mises à jour sont très rapides. Le plus souvent une application lit juste une partie d'un document. Avec l'interface (*DOM : Document Object Model*), seules les parties d'un document qui doivent être chargées en mémoire sont lues par le système de gestion de bases de données. Seules les données nécessaires sont affectées par une mise à jour : il n'y a aucun besoin de lire le document entier dans la mémoire, ni de remplacer la version ancienne.

### 1.6.3 Les caractéristiques attendues d'un SGBD XML et comparaison entre les approches

Un système de gestion de bases de données XML (*XML DBMS*) doit avoir toutes les propriétés standard d'un système de gestion de bases de données, y compris les transactions, la simultanéité d'accès, l'intégrité de données, un processeur de requêtes efficace, etc. Ce qui fait d'un système de gestion de base de données XML un système important pour des applications XML est son efficacité. Un certain nombre de caractéristiques du système de gestion de bases de données doivent être considérées pour la manipulation efficace de XML :

- capacité de comprendre et de traiter XML comme des données.
- support de toute la syntaxe de XML.
- capacité de manipuler efficacement des documents, des fragments et des éléments.
- capacité de rechercher efficacement des documents XML.
- support de (*native XML-based APIS*).
- conservation de tout le contenu de document.
- soutient de (*round-tripping*) de document.
- intégration de données.

L'approche relationnelle est probablement la plus mauvaise, parce qu'elle ne prend pas en compte toute la syntaxe de XML. Une base de données relationnelle est utile seulement pour les documents dit *data-centric*. L'utilisation d'un système relationnel pour stocker XML n'est pas efficace, excepté pour une classe très petite de problèmes qui utilisent XML.

Une approche basée sur le texte est meilleure, parce qu'elle utilise pleinement les possibilités de XML. Cependant, la fonctionnalité principale est perdue dans les secteurs de l'exécution, où les données doivent être constamment réanalysées dans un arbre DOM. De plus, les possibilités de mise à jour sont faibles. En conclusion, l'application automatisée de contraintes d'intégrité crée des risques et des coûts croissants.

L'approche objet peut être considérée comme la meilleure parmi celles considérées : l'accès aux données et la mise à jour sont efficaces, elle permet le traitement de fragments, et elle offre un support total pour la syntaxe de XML et l'intégration.

La table 1.1 suivante récapitule les caractéristiques des trois approches :

| Critère | Base relationnelle | Base Texte | Base Objet |
|---|---|---|---|
| Capacité de comprendre /traiter les données XML | oui | non | oui |
| Support pour toute la syntaxe de XML | non | oui | oui |
| Capacité de manipulation /fragments des documents | certaines | exécution basse | oui |
| Rechercher des documents XML | inefficace | efficace | efficace |
| Support *native XML based APIS* | inefficace | inefficace | efficace |
| Conservation de tout le contenu | non | oui | oui |
| *Round-tripping* | non | oui | oui |
| Intégration de données | oui | non | oui |

TAB. 1.1 – Comparaison entre les différentes approches

## 1.7 Techniques d'Optimisation de requêtes sur des bases de données XML

### 1.7.1 Introduction

Des nombreuses techniques d'optimisation de requêtes sur des données XML ont été découvertes et mises en œuvre par les chercheurs. Parmi elles on peut citer : *path traversal*, utilisation d'indexes, utilisation de vues matérialisées, optimisation basée sur le schéma, reformulation des contraintes XML, *tree pattern matching*, et *labeling scheme*. Dans la section suivante on s'intresse à la méthode d'utilisation d'indexes pour l'optimisation de requêtes sur des bases de données XML, on présente un panorama des approches existantes dans ce domaine.

### 1.7.2 Utilisation d'indexes pour l'optimisation de requêtes

Dans les bases de données XML, la taille d'un document XML est souvent très grande et les DTDs sont souvent hétérogènes, En conséquence, les données XML doivent être organisées d'une manière efficace pour faciliter leur récupération. Pour atteindre ce but on a besoin des techniques pour l'indexation et la récupération des données XML afin de permettre d'accèder facilement à certaines parties des documents recherchés. Ces techniques réduisent la portion de l'arbre XML qui doît être examinée (scannée) pendant le traitement des requêtes. Parmi ces techniques déjà existantes on peut citer DataGuide [Goldman & Widom, 1997], T-index [Milo & Suciu, 1999], Index Fabric [Cooper et al., 2001], BiteCube [Yoon et al., 2001] et XyIndex [Aguilera, 2001].

#### Index Fabric

Les requêtes parcourent les données semi-structurées par l'intermédiaire des expressions de chemins. Elles peuvent être accélérées en utilisant un index. La solution proposée ici code les chemins comme des chaînes de caractères et insère ces chaînes dans un index spécial qui est fortement optimisé pour des clefs longues et complexes. L'*index Fabric* est une structure d'indexation qui fournit l'efficacité et la flexibité dont nous avons besoin. Les structures de données de l'*Index Fabric* sont basées sur des *tries*. Une *trie* est un arbre qui utilise des parties de la clé pour naviguer dans la recherche. Chaque clé est une séquence de caractères, et une *trie* est organisée autour de ces caractères plutôt que toutes les clés. *Patricia trie* (*PT : Patricia Algo-*

*rithm To Retrives information Coded In Alphanumeric*) est une simple forme d'une compressed trie qui fusionne les nœuds enfants avec leurs parents. Chaque chemin de données (*data path*) est codé en utilisant des caractères spéciaux (*designators*) commençant par l'élément racine du document XML. Par exemple, pour le document XML suivant :

```
<library>
    <book id= "B001-05">
      <category title="Computer Concepts 2005">
        <author>Gary Smith</author>
        <author>Robet Wood</author>
        <publisher>Thomson Learning</publisher>
      </category>
   </book>
<library>
```

**library** est représenté par **L**, **B** pour **book**, **C** pour **category** et ainsi de suite. Donc le chemin **library/book/category** est codé par **LBC**. *PT* peut devenir une structure grande et déséquilibrée, ce qui se traduit par une dégradation importante des performances. Pour équilibrer la *trie* et soutenir les données XML de grande taille, [Cooper et al., 2001] ont proposé d'établir de multiples couches de tries et d'effectuer des recherches en procédant d'une couche à l'autre. L'*Index Fabrice* ne contient pas les relations parents-enfants parmis les éléments parce qu'il ne garde pas les informations des éléments XML qui n'ont pas de valeurs atomiques. En conséquence, il est inefficace pour le traitement des requêtes partielles.

### 1.7.3 Optimisation sémantique de requêtes à travers des vues

#### Approche de réécriture de requête en XQuery

[Billel, 2005] propose une technique d'optimisation de requêtes à travers des vues pour le langage XQuery au niveau logique. Cette proposition se base sur un modèle appelé Graphe Structurel de Vues (GSV). Elle permet de reformuler des requêtes à travers des vues en simples requêtes directes sur les documents sources, en ne gardant si possible de l'expression de la vue que les éléments de traitement nécessaires à l'évaluation de la requête de l'utilisateur. Le temps de réponse de la requête reformulée est au pire égal à celui de la requête initiale. L'algorithme proposé permet de réaliser cette optimisation pour des expressions *FLWOR* (*For-Let-Where-Return*) simples.

Dans notre travail, où nous nous intéressons à l'optimisation sémantique de requêtes, nous utilisons la structure d'indexation (ISDs) offerte par Osiris (voir chapitre 4) afin d'effectuer l'optimisation sémantique des requêtes. Cela permet au système de déterminer automatiquement à quelles vues appartient un document XML. En conséquence, l'espace de recherche pour les documents qui font partie de la réponse à la requête peut être réduit à l'espace des vues satisfaisant la requête. Nous expliquons notre méthode en détail dans le chapitre 5.

## 1.8 Conclusion

Dans ce chapitre nous avons passé en revue un ensemble important de concepts, approches et systèmes qui sont en relation directe avec notre projet de recherche.

Après avoir étudié les modèles de données, nous avons exposé les formalismes de représentation existants et nous nous somme concentrés sur les données XML, puis une étude a été faite sur les langages d'interrogation dédiés aux données XML.

Dans le chapitre suivant, nous focalisons notre étude sur la notion d'intégration de données hétérogènes et en particulier les données XML.

# Chapitre 2

# Intégration de Données Hétérogènes

## 2.1 Introduction

Le rôle principal d'un système d'intégration de données est de présenter à l'utilisateur une vue unifiée de données provenant de sources hétérogènes [Dittrich & Jonscher, 2000]. Cette vue unifiée est appelée le schéma global du système. Les approches essentielles utilisées pour définir le schéma global d'un système d'intégration sont *Global As View* (*GAV*) et *Local As View* (*LAV*). Dans l'approche *LAV* les éléments d'un schéma local sont définis comme des vues sur le schéma global. A l'inverse, l'approche *GAV* décrit les éléments d'un schéma global comme des vues sur les schémas de sources locales. Chaque approche a des avantages et des inconvénients. Le traitement de requêtes est plus facile avec *GAV*. En revanche, l'addition d'une nouvelle source est plus facile avec *LAV*.

En général, il y a deux approches utilisées pour la construction d'un système d'intégration. La première est appelée *médiateur*. Dans cette approche, les données réelles existent dans les sources et le schéma global fournit une vue virtuelle cohérente qui permet l'intégration des données dans les sources sous-jacentes. La deuxième approche est l'entrepôt de données. Dans cette approche, les vues sont matérialisées.
Dans le chapitre suivant, nous présentons la notion de vue dans les différents modèles de bases de données (relationnelles, orientées-objet et les données semi-structurées). Nous nous intéressons plus précisément aux modèles de vues existants pour l'intégration de sources de données XML.

Ce chapitre est organisé comme suit. Dans la section 2.2 nous traitons de l'hétérogénéité des données. Dans la section 2.3 nous citons les choix possibles du modèle pivot. Dans la section 2.4 nous détaillons les différentes approches utilisées pour l'intégration de données. Dans la section 2.5 nous étudions le traitement de requêtes dans les différentes approches. Dans la section 2.6 nous présentons des systèmes d'intégration basés sur XML.

Dans la section suivante on s'intéresse aux trois types d'hétérogénéité : l'hétérogénéité structurelle, aussi nommée hétérogénéité schématique, l'hétérogénéité sémantique, et l'hétérogénéité des modèles de données.

## 2.2 L'hétérogénéité des données

L'identification de données dans des bases différentes qui ont des liens sémantiques, ou qui sont sémantiquement liées, consiste en premier lieu à résoudre l'hétérogénéité sémantique et en second lieu à résoudre les différences structurelles (schématiques) [Solar & Doucet, 2002].

Les données utilisées dans notre cadre de recherche sont caractérisées par une hétérogénéité sémantique résultant de l'existence de plusieurs points de vue pour la présentation de données.

Des taxonomies de différences schématiques ont été proposées dans [Breitbart et al., 1986], [Czejdo et al., 1987] et [Kim & Seo, 1991]. Or, les considérations purement schématiques sont insuffisantes lorsque des similarités sémantiques sont à déterminer. Dans la suite, on explique les trois types d'hétérogénéité : schématique, sémantique, et l'hétérogénéité des modèles de données.

**Hétérogénéité schématique**

Quand des concepts équivalents sont modélisés différemment dans des sources de données locales, les conflits schématiques entre les données surviennent. Ces conflits sont associés aux noms, types de données, attributs, et unités. Par exemple, un attribut peut être absent dans l'une des deux entités équivalentes appartenant à des schémas locaux différents. On note que plus le modèle de données est riche, plus le risque de conflit est grand ; par exemple, dans une source on peut représenter l'adresse d'une personne par une chaîne de caractère et dans une autre source par une structure (numéro, rue, ville, code postal). Afin de masquer ces différences, le système d'intégration de données doit construire un schéma global sur un modèle suffisamment expressif.

**Hétérogénéité sémantique**

Ce type d'hétérogénéité survient lorsqu'il y a des différences de signification, d'interprétation ou d'utilisation du même concept (donnée) dans les différents sources [Sheth & Larson, 1990].

La distinction entre l'hétérogénéité sémantique et structurelle (schématique) est importante. En fait, les différences schématiques n'ont intérêt que s'il existe des similarités sémantiques entre des objets. D'un point de vue plus général [Diallo, 2006], traiter le problème de l'hétérogénéité sémantique revient à traiter le problème de la correspondance entre schémas. Dans le domaine de l'intégration c'est un problème crucial, d'autant plus qu'intégrer des sources différentes nécessite l'identification des éléments qui peuvent être liés entre les différents schémas. On parle alors de correspondances (ou *mapping*) entre schémas [Rahm & Bernstein, 2001], par exemple le mapping entre schéma local et schéma global dans notre cas (voir chapitre 5).

Étant donné un concept du monde réel, par exemple le concept Personne, d'après [Solar & Doucet, 2002] on peut distinguer quatre types de relations sémantiques entre deux représentations R1 et R2 de ce concept :

1. R1 identique à R2 : lorsqu'on représente le même concept en utilisant les mêmes constructeurs ;
2. R1 équivalente à R2 : lorqu'on représente le même concept en utilisant des constructeurs différents ;
3. R1 compatible avec R2 : si elles ne sont ni identiques et ni équivalentes ;
4. R1 incompatible avec R2 : si elles sont contradictoires, à cause de différences fondamentales dans la représentation de l'information sous-jacente.

La même information peut avoir différentes compréhensions étant donné la diversité des aspects géographiques et organisationnels. Par exemple, le "Prix" peut être vu en Euros ou en Dollars selon l'aspect géographique (Europe ou Etats-Unis).

**Hétérogénéité des modèles de données**

Le problème ici est la transformation des structures de données et des opérations d'un SGBD dans les structures de données et opérations d'un autre SGBD. La plupart des travaux sur l'intégration supposent que les schémas initiaux sont tous exprimés dans un même modèle de données, appelé modèle commun ou modèle pivot, sur lequel le logiciel d'intégration est construit. Une phase de pré-traitement doit traduire les schémas sources qui ne sont pas exprimés dans ce modèle. En général, ces traductions ne peuvent pas se faire de manière automatique.

L'un des points importants est le choix du modèle de données commun ou pivot (XML dans notre cas). Dans la section suivante on présente des modèles pivots les plus utilisés pour la définition du schéma global.

## 2.3 Choix du modèle pivot

Le modèle pivot est utilisé pour la représentation des données provenant de sources hétérogènes. Dans ce contexte deux critères sont essentiels : la capacité à représenter directement l'ensemble des formats de données, et la flexibilité, pour pouvoir s'adapter facilement aux changements survenant inévitablement dans un contexte hétérogène et distribué.

### 2.3.1 Modèle pivot structuré

Il existe deux grands modèles pivots structurés : le modèle relationnel et le modèle objet. Le modèle pivot structuré doit offrir le plus grand nombre de constructeurs de types possibles afin de pouvoir représenter directement le plus grand nombre de formats possibles. Aujourd'hui, la majorité des chercheurs considèrent que le modèle objet est un meilleur candidat que le modèle relationnel car il dispose de structures plus riches. Mais le problème est que plus un modèle pivot est riche en concepts, plus le processus d'intégration devient complexe puisqu'il peut intégrer les nombreuses divergences dues aux différents choix de modélisation possible du même concept. A noter que lorsque le modèle pivot est moins riche, les contraintes des schémas sources qui peuvent être prises en compte se réduisent considérablement. De plus, dès le départ on doit fournir un schéma d'intégration, ce qui présente deux inconvénients [Siméon, 1999] :

1. obliger les adaptateurs à exporter leurs informations par rapport à ce schéma.
2. réduire la flexibilité du système parce que chaque fois que les sources ou l'application doivent évoluer, les adaptateurs doivent être modifiés.

### 2.3.2 Modèle pivot semi-structuré

Ce modèle offre beaucoup plus de souplesse parce qu' une donnée quelle qu'elle soit est valide. De plus, toute donnée peut toujours être représentée sous forme d'arbre ou de graphe. Par contre, la vérification ou l'optimisation est difficile à cause de l'absence de typage. Comme on l'a souligné dans le premier chapitre, OEM (*objet Exchange Model*) fut le premier modèle de données semi-structurées conçu pour le Web. Il a donc été le premier modèle utilisé pour représenter le schéma pivot. Ensuite, XML s'est imposé comme un nouveau standard de structuration et d'échange de documents semi-structurés. Dans la suite (section 2.7) on montrera les avantages de XML pour la représentation du modèle pivot.

## 2.4 Les différentes approches utilisées pour l'intégration de données

Dans notre projet de recherche, les données de la source sont transformées et stockées dans des tables afin de pouvoir les interroger directement et sans avoir besoin d'accéder aux sources pour les chercher. Cela conduit à l'intégration selon l'approche "Entrepôt de données".
Aujourd'hui, les technologies d'intégration de données prennent deux directions distinctes. La première est celle de **l'entrepôt de données**, contenant une extraction sélective des informations pertinentes stockées dans diverses sources. Suite à la constitution d'un entrepôt, l'utilisateur peut poser ses requêtes sur une base unique, l'entrepôt. La deuxième solution est celle du **médiateur**, qui consiste à fonder l'intégration de données sur l'exploitation de vues abstraites décrivant le contenu des différentes sources d'informations. Les données ne sont pas stockées au niveau du médiateur et ne sont accessibles qu'à partir des sources de données. L'approche entrepôt est utile si les modifications dans les sources locales ne sont pas fréquentes et si le temps de réponse aux requêtes doit être très court. Par contre, l'approche médiateur est préférable si les sources locales sont fréquemment sujettes à des mises-à-jour. Dans cette section, nous présentons en détail ces deux approches avec leurs architectures respectives.

### 2.4.1 L'approche entrepôt de données

Le principe de l'entrepôt de données (*Data Warehouse*) est d'extraire les données jugées utiles, qui sont réparties dans de multiples sources hétérogènes, de les combiner et de les regrouper dans un entrepôt de données. L'alimentation de l'entrepôt de données est faite périodiquement et en mode batch. Lors de l'alimentation, des composants logiciels extraient et nettoient les données avant de les combiner avec des données issues d'autres sources de données, et enfin d'injecter dans l'entrepôt le résultat de ce travail. Cette solution améliore donc en général la qualité des données. Par ailleurs, lorsque l'entrepôt est dédié à l'aide à la décision, il permet à des *end-users* de rechercher eux-même les données nécessaires à une analyse. La cohérence des données est assurée globalement, basée par exemple sur le point de vue du directeur qui cherche des données complètes et sans contradiction. L'alimentation de l'entrepôt en données se réalise en mode batch à travers un module (*ETC : Extraction Transformation Chargement*) qui permet l'extraction, la transformation et le chargement des données [Wu & Buchmann, 1997]. Les entrepôts de données sont souvent choisis dans l'industrie comme le support d'aide à la décision, notamment grâce à leur association aux techniques OLAP [Agarwal et al., 1996]. La figure 2.1 représente l'architecture d'un entrepôt de données [Coulet, 2008].

L'intégration dans l'approche entrepôt de données s'appuie sur un schéma global défini pour l'entrepôt. Des *extracteurs* vont extraire les données des sources et ils les transforment en un format de représentation compatible avec le schéma de l'entrepôt, puis ils les insèrent dans l'entrepôt. En utilisant les techniques classiques d'interrogation de base de données, l'utilisateur peut poser ses requêtes sur le schéma de l'entrepôt. L'utilisateur peut également interagir avec l'entrepôt par l'intermédiaire des *magasins de données* (data marts), qui proposent des vues particulières sur les données afin de faciliter leur analyse dans un processus d'aide à la décision.

Les principaux avantages d'un entrepôt de données sont [Coulet, 2008] :

1. les données sont regroupées dans une seule source, ce qui facilite l'exploitation du système.
2. les données intégrées sont facilement trnsformées et directement adaptées puisqu'elles sont à disposition au sein de l'entrepôt spécifiquement créé.

En revanche, la limite principale à cette approche réside dans le développement nécessaire

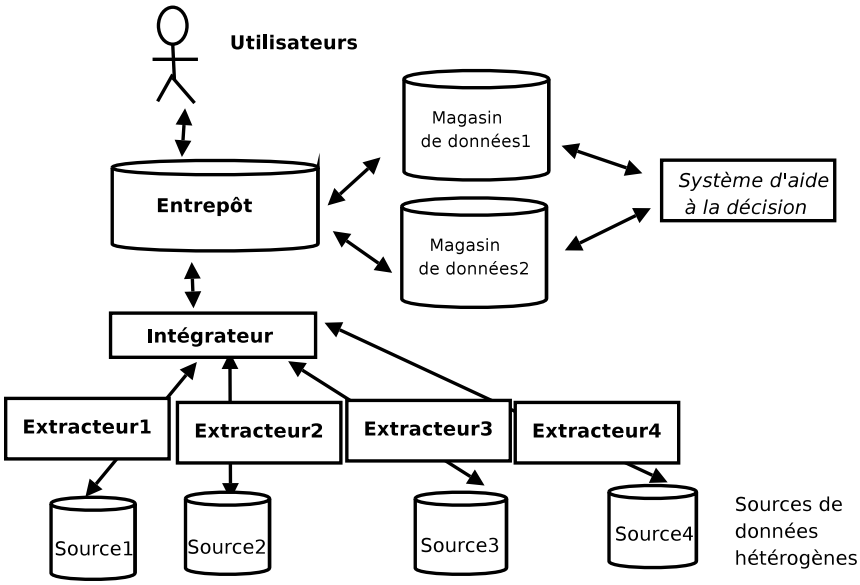


FIG. 2.1 – Architecture de l'approche entrepôt de données

de méthodes d'extraction et d'intégration des données capable de rafraîchir périodiquement le contenu de l'entrepôt, tout en tenant compte de la mise à jour des sources.

### 2.4.2 L'approche médiateur

Ces dernières années, plusieurs systèmes d'intégration de données (par exemple, Information Manifold [Kirk et al., 1995], [Levy et al., 1996], PICSEL [Rousset et al., 2002] et SIMS [Arens et al., 1994]) ont été basés sur une architecture de médiateur [Widerhold, 1992] montrée dans la figure 2.2 [Diallo, 2006], qui fournit une interface uniforme pour poser des requêtes sur des sources de données multiples et hétérogènes. L'utilisateur pose des requêtes en termes de l'ensemble de relations qui ont été conçues pour capturer la sémantique d'un domaine d'application donné (par exemple, le tourisme).
Ces relations sont virtuelles, dans le sens où leurs données ne sont pas directement disponibles mais sont entreposées (stockées) dans les sources. Par conséquent, pour répondre à une requête il faut d'abord traduire la requête de l'utilisateur en une suite de requêtes sur les sources appropriées ; ensuite, une fois la requête exécutée sur chacune des sources, il faut procéder à la fusion des données pour que celles-ci soient au format attendu par l'utilisateur. C'est pourquoi, dans ces systèmes, il y a le besoin d'un ensemble de descriptions des sources. Classiquement, les sources sont décrites par un ensemble de vues, pour lesquelles des contraintes logiques et des correspondances (*mappings*) logiques avec des relations de base source sont spécifiées.

L'avantage le plus important d'un médiateur est qu'il permet aux utilisateurs de se concentrer sur la spécificité de leur demande, en les libérant du besoin de trouver les sources appropriées et de combiner des données des sources multiples pour obtenir des réponses. Ainsi, les utilisateurs ne sont pas obligés de connaître quelles sources sont disponibles. C'est le rôle du médiateur de prendre le contrôle de la construction des plans de requêtes spécialisées (exprimées en termes de vues) et les exécuter afin de répondre aux requêtes originales (exprimées en termes de modèle de

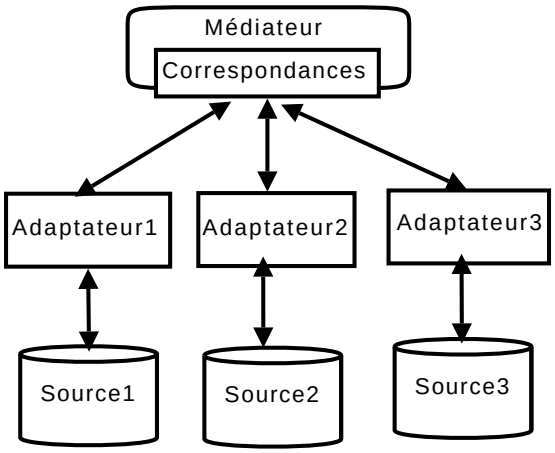


FIG. 2.2 – Architecture d'un médiateur

domaine).

Les différents systèmes d'intégration d'informations à base de médiateurs se distinguent par [Solar & Doucet, 2002] : d'une part, la façon dont est établie la correspondance entre le schéma global et les schémas des sources de données à intégrer, et d'autre part les langages utilisés pour modéliser le schéma global, les schémas des sources de données à intégrer et les requêtes des utilisateurs.

Concernant le premier point, il existe différentes approches pour préciser la correspondance (*mapping*) entre le schéma global et les sources locales. Ces approches sont :

- L'approche GAV (*global-as-view*) [Batini et al., 1986] [Halevy, 2001] : Les éléments d'un schéma global sont décrits comme des vues (virtuelles) sur les schémas de sources locales. Ces définitions de vues sont utilisées pour réécrire des requêtes qui sont définies sur un schéma global sous la forme de requêtes distribuées sur les bases de données locales. Des exemples de l'approche GAV sont TSIMMIS [Chawathe et al., 1994], InterViso [Templeton et al., 1995] et Garlic [M.Roth & Schwarz, 1997]. L'inconvénient principal de GAV est qu'il n'est pas facile d'assurer l'évolution des schémas locaux.

- L'approche LAV (*local-as-view*) [Batini et al., 1986] [Halevy, 2001] : Les éléments d'un schéma local sont définis comme des vues (matérialisées) sur le schéma global, et le traitement des requêtes sur le schéma global implique la réécriture de requêtes en utilisant des vues. Des exemples de l'approche LAV sont IM (*Information Manifold*) [Levy et al., 1996] et Agora [Manolescu et al., 2001]. LAV isole les modifications de schémas locaux pour qu'elles n'affectent que la dérivation de règles définies pour ce schéma. Mais il y a des problèmes si on a besoin de changer le schéma global, puisque toutes les règles pour définir les schémas locaux comme des vues du schéma global devront être revues.

- Les approche mixtes : Récemment, de nouvelles approches combinant des caractéristiques GAV et LAV sont apparues :

  1. L'approche BAV (*both-as-view*) : [Brien & Poulovassilis, 2003], [Boyd et al., 2004],

spécifie un *mapping* bidirectionnel entre chaque source et le schéma global. Ce mapping bidirectionnel permet aux données et aux requêtes d'être traduites dans les deux sens, du schéma global aux sources et vice-versa. Ceci est important, par exemple, lors de l'intégration de données dans les contextes peer-to-peer [McBrien & Poulovassilis, 2003].

2. L'Approche GLAV (*Global Local View*) est une variante de LAV qui permet d'avoir comme définition de la vue n'importe quelle requête sur le schéma local [Ullman, 2000], [Sattler et al., 2005], [Deutsch & Tannen, 2003].

3. L'Approche BGLAV (*BYU-Global-Local-as-View*) est une approche qui se base sur des outils de génération automatique de *mapping*, et qui permet de basculer en mode LAV ou GAV selon que l'on est dans une étape d'ajout de nouvelles sources ou de traitement de requête [Xu & Embley, 2004].

L'approche suivie dans notre projet de thèse est la première approche, *global-as-view* (GAV), qui facilite le traitement de requête sur le schéma global de l'entrepôt de données.

Les systèmes existants se différencient également par le langage qu'ils utilisent pour exprimer le schéma global. On distingue les systèmes suivants :

- Les systèmes fondés sur un schéma global à base de règles (Infomaster, Information Mainfold) ;
- Les systèmes fondés sur un schéma à base de classes (TSIMMIS), logique de description (SIMS, OBSERVER),ou encore des systèmes combinant le pouvoir d'expression d'un formalisme à base de règles et d'un formalisme à base de classes (PICSEL) ;
- Les systèmes médiateurs au-dessus de données semi-structurées ayant le format de documents XML (Xylème [Cluet et al., 2001]). Ils sont fondés sur un schéma global à base d'arbres. Ils relèvent à la fois de l'approche GAV et LAV, la correspondance entre le vocabulaire du médiateur et celui des sources étant exprimée par de simples *mappings* des chemins.

## 2.5 Traitement de requêtes

La suite est une description et comparaison du traitement de requête dans les approches d'intégration de données.

### 2.5.1 Dans l'approche médiateur

Comme on l'a expliqué dans la section médiateur, l' utilisateur pose des requêtes sur le schéma global ou virtuel, qui fournit un vocabulaire unique, et non sur les schémas locaux. Pour pouvoir répondre à ces requêtes et parce que les données sont distribuées dans les sources, le système utilise des *wrappers* (adaptateurs) pour effectuer la réécriture ou la traduction de la requête utilisateur exprimée dans les termes du vocabulaire du schéma global en des requêtes exprimées selon le vocabulaire des sources locales. Ensuite, les adaptateurs traduisent les réponses aux requêtes sur les sources en des réponses compatibles avec le schéma global du médiateur. Cette traduction se base sur la définition des mappings entre le schéma global et les schémas locaux. On doit s'assurer que les réponses obtenues par la réécriture seront correctes et complètes pour la requête, autrement dit, le résultat de la réécriture de la requête doit contenir toutes les réponses possibles extraites depuis les sources de données. De plus, les sources qui ne participent pas à la requêtes seront exclues de la réécriture.

On note aussi que, dans l'approche médiateur, la processus de traduction de requêtes diffère aussi selon le choix de mapping entre le schéma global et les schémas locaux. Ainsi, dans l'approche GAV, la traduction de requêtes est un simple processus de dépliage (*unfolding*). Dans le cas de LAV, la requête sur le schéma global doit être reformulée en termes de schémas de sources de données locales. C'est un processus appelé "réécriture de requêtes en utilisant les vues" [Halevy, 2001], [Calvanese et al., 2000].

Étant donné une requête q et un ensemble *V1, ..., Vn* de vues sur les relations du schéma global, est-il possible de répondre à la requête en utilisant uniquement la réponse aux vues.

Le système d'intégration essaie d'effectuer la réécriture des requêtes des utilisateurs en s'assurant que les requêtes sont soit équivalentes aux requêtes initiales, soit incluses (de façon maximale) dans chacune des requêtes initiales [Diallo, 2006]. La réécriture maximale de requête essaie de trouver la meilleure solution possible par rapport aux sources disponibles. L'algorithme *MiniCon* [Pottinger & Halevy, 2001] est jugé le plus performant pour la réécriture de requêtes. Les requêtes et les vues acceptées par MiniCon sont des requêtes conjonctives avec des prédicats de comparaison arithmétique.

Dans [Sebi, 2007] les auteurs ont montré que trouver une réécriture est un problème *NP-Difficile*, pour deux raisons :

1. la charge de tester les *mappings* entre une vue et une requête ;
2. le nombre de possibilités pour combiner les *mappings* des différentes vues par rapport à la requête.

Des techniques de réécriture pour l'optimisation de requêtes sont décrites dans [Adali et al., 1996] et dans [Fernandez & Suciu, 1998] pour le cas de requêtes XPath dans les données semi-structurées.

### 2.5.2 Dans l'approche entrepôt

Dans cette approche, l'utilisateur pose une requête sur le schéma global de l'entrepôt en utilisant les techniques classiques d'interrogation de bases de données. Comme les données sont regroupées dans une seule source dans l'entrepôt, le traitement de requête se fait d'une façon rapide et efficace sans avoir besoin de la phase de traduction. En revanche, l'approche entrepôt est très coûteuse dans le cas où les sources locales sont soumises à une évolution fréquente, parce que cela implique la mise à jour de l'entrepôt à chaque modification des sources, comme par exemple dans les domaines des sciences du vivant, où on a une évolution hebdomadaire, voire quotidienne.

## 2.6 Utilisation des ontologies pour l'intégration de données hétérogènes

La définition intiale d'une ontologie est donnée par Tom Gruber :

"Une ontologie est une spécification explicite d'une conceptualisation". Une ontologie permet de fournir un vocabulaire partagé entre plusieurs utilisateurs. De plus, elle facilite la communication de connaissances. Les domaines des bases de données et de l'intelligence artificielle utilisent de plus en plus le terme d'ontologie. Bien qu'on trouve pas une définition unique du terme ontologie, le rôle des ontologies est clair, elles doivent fournir une description concise et non ambiguë des "Concepts" et de leurs "Relations" pour un domaine d'intérêt.

Dans le domaine de l'intégration de données, les ontologies peuvent contribuer à la résolution des problèmes d'hétérogénéité syntaxique et sémantique. Les ontologies permettent la description formelle des concepts d'un domaine ainsi que des relations entre ces concepts. Grâce à une défnition formelle à laquelle est associée une sémantique clairement établie, un utilisateur peut décrire une donnée ou une relation dans une source. Ensuite, il peut exploiter cette définition pour intégrer le contenu de la source de façon non ambiguë. Dans ce cas, on parle d'approche d'intégration fondée sur une ontologie ou d'intégration sémantique, parce que les définitions formelles et sémantiques peuvent en pratique être représentées sous la forme d'axiomes logiques composant une ontologie. Parmi ces systèmes d'intégration fondés sur une ontologie, on peut citer : SIMS [Arens et al., 1998], PICSEL [Rousset & Reynaud, 2003] et OBSERVER [Mena et al., 2000].

## 2.7 Utilisation de XML pour l'intégration de données hétérogènes

XML est rapidement devenu un standard pour la représentation et l'échange de données. Il fournit un format commun pour exprimer à la fois la structure et le contenu des données. C'est pourquoi il peut être utile pour l'intégration des données structurées, semi structurées, et des données non structurées. Cependant, XML seul ne peut pas fournir la solution complète au problème d'intégration de données [Halevy, 1999] [Widom, 1999] et il reste plusieurs défis auxquels il faut faire face. Dans la suite on explique les rôles que XML peut jouer dans le processus d'intégration.

### 2.7.1 Le rôle de XML dans l'intégration de données

Dans cette section, on montre où et comment XML peut aider le processus de l'intégration de données et où XML n'est pas suffisant. Plus précisement, on parle des rôles de XML dans la réconciliation à différents niveaux : réconciliation de modèles de données, réconciliation de schéma et réconciliation d'instances de données.

#### Réconciliation de modèles de données

XML fournit une manière tout à fait naturelle pour structurer les données, basée sur les représentations hiérarchiques et sous forme de graphes. Un tel modèle a l'avantage d'être simple, standard, et bien admis, mais également assez puissant pour représenter les informations structurées, non structurées et semi-structurées. Ainsi, XML fonctionne bien comme un modèle commun pour les données du Web, et une grande variété d'outils ont été mis sur le marché pour faciliter la manipulation des données de bases relationnelles au format XML. En outre, plusieurs travaux de recherche ont étudié l'utilisation des modèles des données basés sur les représentations sous forme de graphes pour contrôler les données semi-structurées et pour intégrer des sources de données hétérogènes.

#### Réconciliation de schémas

Les noms et les significations des balises (*tags*) de XML sont arbitraires, ce qui rend indispensable la définition d'une norme pour les balises et les schémas pour un domaine spécifié. De nombreux efforts ont été faits pour atteindre cet objectif (par exemple, Biztalk[2] et Oasis [3]). Dans notre projet de thèse on a utilisé XSLT (*Extensible Stylesheet Language Translation*)

[2]Biztalk :*http ://www.biztalk.com/*
[3]Oasis :*http ://www.oasisopen.org/*

[Goh et al., 1999] pour définir des correspondances (*mappings*) entre les balises (*tags*) hétérogènes des documents XML dans la base source de données.

La standardisation des étiquettes et des schémas simplifie considérablement la tâche de l'intégration de données, mais nous avons également besoin de décrire la sémantique des éléments et des attributs.

Une sémantique des informations plus complexe peut exiger des représentations alternatives. Le procédé de réconciliation peut également expliquer des informations sur le contexte, telles que des significations d'un nom particulier ou d'une valeur spécifique, qui dépendent du contexte dans lequel la donnée a été produite. Par exemple, un identificateur numérique d'un employé peut donner l'information appropriée aux lecteurs qui connaissent les conventions de numérotation en place dans l'organisation, c'est-à-dire que l'information et le lecteur partagent le même contexte.

#### Réconciliation d'instances de données

Les méta-données jouent un rôle crucial dans la réconciliation d'instances des données, déterminant comment traiter l'information semblable ou contradictoire. Par exemple, les méta-données permettent d'attacher une marque de temps (*time stamp*) ou un niveau de qualité aux données intégrées, et ces informations peuvent être employées pour résoudre des conflits parmi les informations stockées dans des différentes sources.

### 2.7.2 XML est-il suffisant ?

Bien que XML puisse être utile dans les processus d'intégration de données et réduise le travail de réconciliation des sources de données hétérogènes, il n'est pas suffisant pour résoudre le problème général de l'intégration de données. Il est important pour l'intégration de données de disposer d'une langue pour indiquer la sémantique liée au contenu des données. Le W3C a identifié ce besoin. Les groupes de travail *Resource Description Framework* [4] et Web Sémantique (*Semantic Web*), étudient des questions liées à la représentation des aspects sémantiques des données du Web. L'objectif principal du Web Sémantique est de définir des architectures, des modèles et des standards pour fournir une description lisible par la machine (*machine-readable*) de la sémantique des sources du Web. les principales questions pertinentes pour l'intégration des données sont :

- le développement d'une base formelle pour les standards des données du Web ;
- le développement des techniques et des outils pour la création, l'extraction et le stockage de méta-données ;
- le développement des outils basés sur la sémantique pour la découverte de connaissances.

Le développement des outils appropriés pour l'intégration des sources des données basée sur XML est aussi important. Ces outils doivent permettre l'intégration automatisée autant que possible aux trois niveaux de données : modèle, schéma, et instances.

## 2.8 Systèmes d'intégration basés sur XML

### 2.8.1 TSIMMIS

**TSIMMIS** (*The Stanford-IBM Manager of Multiple Information Sources*) : L'objectif du système TSIMMIS [Hammer et al., 1995] [Papakonstantinou et al., 1995] [Chawathe et al., 1994] est de permettre le développement d'outils qui facilitent l'intégration des sources hétérogènes de

[4] RDF, *http ://www.w3.org/TR/REC-rdf syntax*

données qui contiennent à la fois des données structurées et semi-structurées. Il offre un modèle de données (OEM), qui est un modèle simple, où chaque objet a une étiquette qui informe sur le contenu de ses données. Il simplifie la conversion des données et permet le développement d'un traducteur d'une manière automatique ou semi-automatique.

TSIMMIS utilise un langage commun de requêtes (MSL) pour permettre l'interrogation de données stockées dans des différentes sources. Les composants principaux de TSIMMIS sont :

- un adaptateur (*wrapper*), qui transforme des données et aussi des requêtes au formt du modèle commun d'informations. Il convertit les requêtes OEM-QL sur des données du modèle commun OEM en requêtes que la source peut exécuter ; il convertit également les résultats envoyés par la source en données représentées par le modèle OEM ;
- un médiateur, qui combine les données d'une seule source ou de plusieurs sources ;
- des applications clientes : plusieurs applications clientes ont été développées pour offrir à l'utilisateur final une interface conviviale ; ces applications permettent de naviguer dans les données à travers le web.

Grâce à son architecture, cette approche permet aux nouvelles sources de devenir utiles dès qu'un traducteur sera fourni, et à des médiateurs d'accéder à des nouvelles sources d'une manière transparente. Ces médiateurs peuvent travailler indépendamment. L'inconvénient majeur dans une telle approche est que l'on perd la notion de schéma, de classe et d'extension ; il n'y a personne qui ait une vision globale du système d'information. En conséquence, la classification des données et l'optimisation des requêtes sont impossibles et seules les requêtes prédéfinies sont exécutables.

### 2.8.2 Le système MIX (*Mediation of Information Using XML*)

L'objectif de ce projet est d'utiliser XML afin de développer un système pour l'interrogation de sources de données hétérogènes [Papakonstantinou & Velikhov, 1999] [UCSD & SDSC, 1999]. Pour le groupe du travail qui a bâti MIX, le Web est une énorme base de données répartie et XML est un bon modèle pour représenter les données de cette base. Ils supposaient que dans l'avenir, la plupart des sources de données permettraient d'exporter leurs données au format XML. Cette hypothèse semble justifiée, car de nombreux logiciels permettent aujourd'hui d'exporter leurs données en XML et on parle du Web XML [Laurent et al., 2003]. Les composants principaux de MIX sont :

- le composant *DOM for Virtual XML Doc's*, qui réalise l'interface externe du système ;
- le composant *MIX Mediator*, qui réalise la médiation des données.

Le système accède à un ensemble de sources XML pour fournir une vue XML sur les données des sources. Il dispose aussi d'un ensemble de requêtes auxquelles il peut répondre. Conceptuellement, MIX exporte une vue intégrée (globale) des données de base. Cette vue définit comment les données de base seront intégrées. Cette définition permet de déduire une DTD qui constitue le schéma de la vue. L'ensemble des vues sur les sources définit le schéma médiateur en utilisant le langage XMAS [Papakonstantinou & Vianu, 2000], qui est inspiré de XML-QL [Deutsch et al., 1998]. Noter que MIX ne matérialise pas à l'avance la vue intégrée.

Le composant *DOM for Virtual XML Doc* offre aux utilisateurs des documents XML virtuels en utilisant les DTDs générées par les vues. Ces documents sont présentés sous forme d'arbre, à l'aide du modèle de données DOM. Pendant le temps d'exécution, une application BBQ (*Blended Browsing and Querying*) permet de définir des requêtes sur la vue intégrée en utilisant le langage XMAS. Le médiateur décompose la requête du client en plusieurs requêtes qui sont adressées aux sources XML. Les sources produisent et envoient au médiateur des résultats de requêtes, qui sont des documents XML. Le médiateur les intègre dans le document résultat qui contient

la réponse à la requête de l'utilisateur.

**En résumé**, MIX est basé sur l'architecture bien connue d'un médiateur afin de fournir à l'utilisateur une vision intégrée des sources sous-jacentes. MIX utilise XML pour faciliter une représentation uniforme et souple d'une source de données arbitraire. Dans le cas où les schémas des données sont absents, ce système libère l'utilisateur du problème de formulation des requêtes pertinentes sur les bases de données semi-structurées, en utilisant les DTDs comme une description structurelle des données échangées par les composantes du l'architecture d'un médiateur. Le schéma fourni par une DTD est plus flexible que les schémas relationnels, et en même temps il fournit plus de structure que le modèle semi-structuré complet des approches existantes comme TSIMMIS. Étant donné le rôle central des DTD dans cette approche, l'inférence semi-automatique de vues DTDs devient une question importante. Dans [Papakonstantinou & Velikhov, 1999] un algorithme a été présenté pour l'inférence de DTDs dans le système MIX.

### YaT

YAT [Siméon, 1999] est un système pour l'intégration des sources des données hétérogènes basé sur XML. Il représente les données semi-structurées par des graphes. Le langage YATL utilisé est un langage déclaratif qui permet de spécifier des règles de transformation de données représentées dans un modèle (objet, relationnel ou semi-structuré) en un autre modèle. Il fournit aussi des primitives de restructuration et des fonctions afin de créer des graphes complexes et de traiter des identifiants. Cependant, YATL n'est pas un langage de requêtes, et il ne permet pas d'exprimer des requêtes comme des expressions de chemin généralisées. Grâce à son typage et ses bonnes propriétés de compositionnalité, YaTL facilite la réutilisation de programmes sans perte d'efficacité. Enfin, des techniques d'optimisation des requêtes distribuées ont été développées en utilisant une algèbre XML.

### Conclusion

Dans ce chapitre, nous nous sommes focalisés sur la notion de données hétérogènes, leurs types, les approches utilisées pour leur intégration. Nous somme ensuite passés à l'étude des approches existantes pour le traitement de requêtes. Enfin, le chapitre s'est terminé par une représentation des systèmes d'intégration basés sur XML.

A ce stade on a fait le choix de l'approche qu l'on va utiliser pour l'intégration de données, qui est l'approche *entrepôt de données* et qui s'adapte le mieux pour notre domaine d'étude, parce que notre objectif est d'optimiser le temps de réponse de la requête. La section *traitement de requêtes* a éclairé le processus de traitement des requêtes selon les approches médiateur et entrepôt de données afin que l'on puisse comprendre, dans le chapitre suivant, le principe d'utilistion des vues pour l'intégration de données XML.

# Chapitre 3

# L'utilisation des vues pour l'intégration de données semi-structurées

## 3.1 Introduction

Dans le domaine des bases de données, les vues sont utilisées pour des raisons aussi diverses que restreindre l'accès aux données pour des utilisateurs non autorisés, restructurer des données ou encore offrir une vision unique à des données qui proviennent de sources différentes. Notre travail est concerné par cette dernière situation. Dans ce contexte, les vues permettent d'utiliser une structure unique pour accéder à de grands volumes de données hétérogènes.

Les vues sont donc utiles pour l'intégration de données. Un système d'intégration de données fournit une interface unifiée pour l'interrogation de sources multiples de données hétérogènes, distribuées et autonomes, données qui peuvent être stockées dans des bases de données classiques ou dans des bases de données distribuées. L'idée est de rendre transparent pour l'utilisateur le fait que les données nécessaires à une réponse complète à sa requête sont stockées sur plusieurs sources, en général hétérogènes. Les requêtes sont en général formulées en utilisant un langage unifié d'interrogation sur des vues virtuelles unifiées de données (aussi connu sous le nom de médiateur) au lieu d'être formulées en termes des schémas des diverses sources de données. Le système d'interrogation traduit la requête sur la vue dans des (sous-) requêtes sur les sources et, après le traitement des résultats fournis par les diverses sources, il présente le résultat final à l'utilisateur [Lenzerini, 2002].

## 3.2 Définition d'une vue

Dans le contexte de l'intégration de données, une vue est considérée comme un triplet (domaine, schéma, définition) qui permet de restructurer des données et de fournir un schéma unifié de ces données [Baril, 2003]. La figure 3.1 illustre cette définition en présentant les éléments du concept de vue :

- Le **domaine** est l'ensemble des sources contenant les données de la vue. Sur la figure 3.1, les trois sources qui consta marcheituent le domaine de la vue sont encadrées.
- Le **schéma** est la structure utilisée par la vue pour présenter les données. Ce schéma joue le rôle d'interface entre les utilisateurs de la vue et les données des sources. Dans le contexte relationnel, le schéma de la vue est celui de la requête SQL qui permet de calculer le résultat.

– la **définition** spécifie les données qui seront associées au schéma de la vue. Dans le contexte relationnel, la définition de la vue est une requête SQL.

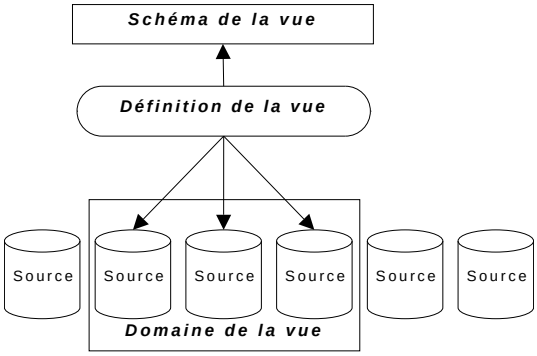


FIG. 3.1 – Définition d'une vue.

Dans l'état de l'art proposé par [Cannataro et al., 2005], les vues dépendent toujours du modèle de base de donnés, qu'il soit relationnel, orienté-objet ou un modèle de données semi-structurées. Nous détaillons cette dépendance dans ce qui suit.

### Dans le modèle relationnel

Dans ce modèle, l'organisation de données se fait dans des tables (relations), qui ont un schéma définissant leur structure. L'instanciation d'une relation contient des n-uplets [Ullman, 1988] ; par exemple, l'ensemble suivant de relations permet de définir des données sur les personnes et les services d'un hôpital :

```
Personne (Id_Personne, Nom, Prenom, Adresse, Salaire);
Service (Num_Serv, Nom_Serv, Chef);
Travail_Dans_Service (Id_Médecin, Id_Service);
```

Pour l'interrogation de données, le langage SQL, qui est un langage standard d'interrogation pour le modèle relationnel, est utilisé en général. Les bases de données peuvent être constituées d'un grand nombre de tables, tandis que l'utilisateur peut n'avoir besoin que d'une petite partie des données. Les vues permettent de créer une relation virtuelle avec des données relatives à un sous-ensemble d'attributs. Elles sont calculées à partir des tables de la base de données.

Par exemple, pour surveiller le médecin qui travaille dans le service de cardiologie (et aussi pour protéger les informations sur le salaire d'un accès non autorisé), on peut créer une vue avec le nom du service et le nom du médecin.
La vue Médecin_de_Service Cardiologie peut être définie par la requête SQL suivante, (cette partie est extraite de l'exemple de l'hôpital, voir annexe B) :

```
CREATE VIEW Médecin_de_Service cardiologie AS
SELECT   Nom AS NomMédecin
         Adresse AS  Adresse_Médecin
         NomServ AS  Nom_Service
FROM Personne, Service, Travaille_Dans_Service
WHERE Nom_Serv= 'cardiologie'
AND   Id_Personnes=Id_Médecin
AND   Num_Service = Id_Service
```

L'utilisateur peut interroger la vue Médecin_de_Service cardiologie comme si c'était une relation de la base de données, en utilisant une expression SQL. la requête suivante fournit les noms des médecins affectés au service de cardiologie.

```
SELECT NomMédecin
FROM   Médecin_de_Service cardiologie
```

Le traitement d'une requête sur une vue dépend de la mise en oeuvre de la vue. Il est dit standard si la vue est traitée comme une relation de la base de données ; dans ce cas, la vue est matérialisée. Dans le deuxième cas, où la vue ne contient pas de données, c'est une vue virtuelle, où les valeurs ne sont pas dupliquées (*replicated*) et représentent donc une source virtuelle que l'utilisateur peut interroger comme si elle était stockée dans la base de données, moyennant en général un calcul de l'extension de la vue lors de l'exécution de la requête.

Par exemple, la requête précédente sur la vue Médecin_de_Service cardiologie est traduite par le système en une requête sur les sources de données actuelles (i.e., les relations Personne, Service et Travaille_Dans_Service ), combinant la définition de la vue avec la requête comme suit :

```
SELECT  NomMédecin
FROM (SELECT Nom AS NomMédecin
             Adresse AS  Adresse_Médecin
             NomServ AS  Nom_Service
      FROM Personne, Service, Travaille_Dans_Service
      WHERE Nom_Serv= 'cardiologie'
      AND   Id_Personnes=Id_Médecin
      AND   Num_Service = Id_Service)
```

Cette requête peut ensuite être évaluée par le processeur de la requête comme une requête standard.

On peut **conclure** qu'une vue dans un système relationnel est créée en sélectionnant :

- un domaine.
- un schéma virtuel.
- les correspondances entre cette vue et les données dans l'ensemble des relations en utilisant une requête.

Des propositions ont été faites pour la définition des vues dans les modèles orienté-objet et semi-structuré.

### Dans le modèle de données orientées-objet

D'après [Cattell et al., 1997], les éléments sont des instances de classes, c'est à dire des objets. Un objet est représenté par un couple (oid, valeur), où **oid** est an identificateur unique et **valeur** est une liste de propriétés primitives (comme integer, string, etc.) ou d' objets complexes.
Un schéma d'une base de données dans le modèle orienté-objet est exprimé par la figure 3.2. Les relations d'héritage sont représentées par des lignes pleines et les relations d'agrégat entre les propriétés par des lignes pointillées

De nouveau, les vues sont définies par un triplet : domaine, schéma et définition.
Le domaine d'une vue est défini en sélectionnant une ou plusieurs classes du schéma de la base de données. Le schéma d'une vue est une nouvelle classe appelée classe virtuelle, et le langage de

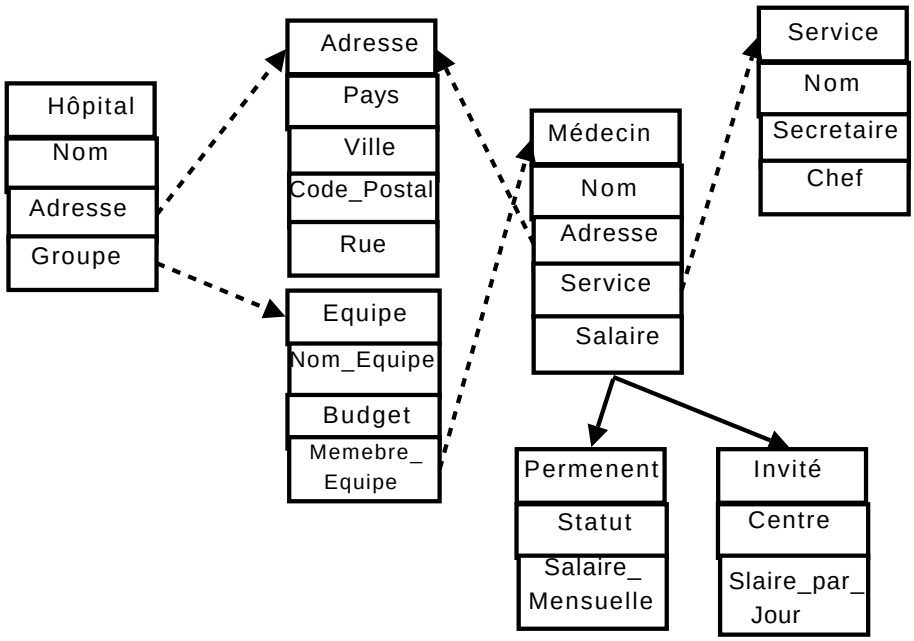


FIG. 3.2 – schéma d'une base orientée - objet

définition d'une vue est en général une extension de *OQL*, le langage standard d'interrogation pour les bases de données orientées-objet. OQL permet la construction de nouveaux objets au travers d'une requête. Par exemple, la classe virtuelle pour des médecins qui travaillent dans le service de cardiologie est définie comme suit :

```
virtual class Médecin_de_Service cardiologie
  hide attributes salaire
  with extension Médecin_de_Service cardiologie =
  select e from Médecin
   where e.Service.Nom="cardiologie"
end;
```

où la notation "." est utilisée pour naviguer dans les associations entre les classes. Grâce à l'héritage, la définition d'une vue sur le médecin permet l'accès à des objets de ses sous-classes, i.e., les classes *Permanent* et *Invité*. Il est possible de limiter l'accès aux sous-classes dans une vue en utilisant la commande *"hide class"*.

### Dans le modèle des données semi-structurées

Ce modèle est basé sur la conception des instances de données comme un graphe, avec des objets et des étiquettes, où les objets sont les sommets du graphe. Les entités sont des objets, et chaque objet a un identificateur unique, appelé *oid*.
Certains objets sont atomiques et contiennent des valeurs d'un type de base (integer, string, etc.), d'autres objets sont complexes, Ils sont définis par un couple (étiquette, oid) [Abiteboul et al., 2000], [Abiteboul et al., 1995].

Les vues dans un modèle de données semi-structurées peuvent être définies en adaptant les données aux besoins de l'utilisateur et avec l'objectif d'augmenter la flexibilité d'un système de base de données.

La vue est un sous-graphe du graphe de la base de données semi-structurée, et les objets dans la vue (i.e., les nœuds ) peuvent être une copie des objets de la source (i.e., matérialisés), ou ils peuvent référencer les autres objets existant dans la base de données. Encore une fois, dans ce cas, les requêtes sur les vues sont reécrites en utilisant la définition de la vue.

XML est utilisé pour définir des vues afin d'exporter des données au format des vues XML sur le Web tout en conservant leur structure interne de données. Les vues XML sont en général interrogées en utilisant *Xquery*, le langage d'interrogation standard pour XML, et ensuite la requête XQuery est traduite dans le langage d'interrogation implémenté par les modèles propres aux sources de données. Les vues XML peuvent aussi être utilisées comme une vue globale pour l'intégration des bases de données.

## 3.3 Les vues virtuelles et matérialisées

Dans un *SGBD*, il existe deux stratégies pour gérer les vues [Baril, 2003] :
dans la première, on stocke seulement la définition de la vue, et on parle alors de vue virtuelle, tandis que dans la deuxième on calcule et on stocke le résultat de la vue ; on parle alors de vue matérialisée.

Lorsqu'une vue est matérialisée, son résultat est calculé puis stocké. L'avantage des vues matérialisées est qu'elles permettent d'accéder rapidement aux données de la vue, car celles-ci sont stockées. Lorsque les vues matérialisées sont utilisées pour répondre à des requêtes, la phase de réécriture continue d'être nécessaire, mais le traitement des requêtes est accéléré car il n'y a pas besoin de calculer son extension.

L'inconvénient des vues matérialisées est qu'il faut maintenir à jour les données de la vue. En effet, lorsque les données des sources sont modifiées, les données de la vue matérialisée doivent être modifiées en conséquence pour rester cohérentes avec la définition de la vue. Pour cela, et dans certains cas simples, on peut utiliser des techniques de maintenance incrémentale qui permettent de propager les modifications et de recalculer seulement les données du résultat qui sont affectées par la modification.

Lorsqu'une vue est virtuelle, son extension n'est pas stockée. Elle doit donc être recalculée à chaque accès à la vue. L'avantage de cette technique est qu'elle évite les problèmes de maintenance des données. Cependant, la phase de calcul des données peut être coûteuse lorsqu'on veut accéder à l'extension de la vue.

Le Système COMMIX [WANG et al., 2004], propose une méthodologie permettant de choisir les vues à matérialiser. D'une manière générale, on doit choisir les vues qui améliorent la performance du système d'intégration de données, c'est à dire les vues telles que le temps d'évaluation d'une requête ou le coût de la maintenance des vues soit minimisé. Cette méthodologie est basée sur la connaissance des requêtes les plus demandées et sur l'algorithme HAVS ( *Heuristics Algorithm for Generating Materialized XML Views*). Etant donné un ensemble de requêtes Q1,Q2,...,Qk les plus attendues pour un système d'intégration, la méthodologie de COMMIX est la suivante :

1. on analyse les requêtes XML les plus demandées et on construit un graphe pour chacune de ces requêtes.

2. dans un deuxième temps, COMMIX fusionne les graphes des requêtes de façon incrémentale pour générer le plan global de requêtes, Les nœuds dans ce graphe sont des vues potentiellement matérialisées. Des règles d'inférence permettent de retrouver des patrons

(*patterns*) communs entre le graphe qui intègre les i premières requêtes déjà traitées et le graphe de la i+1 ième requête.

3. enfin, COMMIX utilise l'algorithme HAVS pour, à partir du graphe global, aussi nommé plan global, choisir l'ensemble des vues XML à matérialiser.

Etant donné que dans notre cas d'étude (l'approche OSIX), on stocke les données elles-mêmes et pas seulement la définition de la vue, les vues sont matérialisées.

## 3.4 Modèles de vues existants pour l'intégration et l'interrogation de sources de données XML

### 3.4.1 le système *Xyleme*

Dans cette section on présente une proposition consistant à utiliser les vues pour l'accès aux documents XML hétérogènes et à grande échelle. Cette proposition a été présentée dans [Cluet et al., 2001], et implémentée dans le système Xyleme [Xyleme, 2001], qui a été lancé en Septembre 1999 à l'initiative du groupe VERSO à l'INRIA. Ses objectifs étaient :

- la construction dynamique d'entrepôts de données semi-structurées ;
- de fournir des solution de recherche sur une grande masse d'information au format XML ;
- de fournir un accès intelligent à une source d'information ;
- de permettre à l'utilisateur de formuler une requête sur un schéma unique.

Xyleme permet de poser des questions précises (grâce à l'utilisation de la structure des documents) et d'obtenir des informations exactes.

Un des principaux défis du projet Xyleme était la résolution du problème d'intégration sémantique de données qui sont hétérogènes et distribuées dans différentes sources. Pour répondre à ce défi, XYLEME propose un mécanisme de vues qui permet à l'utilisateur de formuler une requête sur une structure unique, qui représente un domaine spécifié, plutôt que sur plusieurs structures hétérogènes de documents stockés dans les différentes sources.

#### Les vues dans Xyleme

Dans Xyleme, afin de distribuer au mieux les documents sur l'ensemble des machines de stockage tout en assurant un regroupement par proximité sémantique, le module de classification sémantique classe les documents par *cluster*. Par exemple, le *cluster* **art** contient les documents instances de DTDs relatives à la thématique de l'art. La définition d'une vue en Xyleme est fortement inspirée de la notion de vue en relationnel (voir figure 3.3). Une vue est composée d'un domaine, d'un schéma et d'une requête. Le domaine d'une vue est un sous-ensemble de sources de données, tandis que le schéma d'une vue est une DTD ou un arbre abstrait qui décrit des documents abstraits, c'est à dire des documents virtuels de la vue. On note que tout ce qui est relatif au domaine de la vue est appelé concret (les données dans leur implémentation réelle) et tout ce qui est relatif aux vues elles-mêmes est appelé abstrait.

Dans le médiateur de Xyleme, le schéma global est un groupe de DTDs abstraites, tel que chaque DTD abstraite représente un schéma médiateur pour les données relatives à un même domaine (tourisme, art, biologie, ...). Ainsi, le schéma médiateur est représenté sous la forme d'une hiérarchie de termes pertinents pour un domaine spécifique. La structure de chacun des documents se trouve dans un *cluster* et est décrite par une DTD concrète. Un exemple d'une vue sur le domaine de la culture est montré dans la figure 3.4.

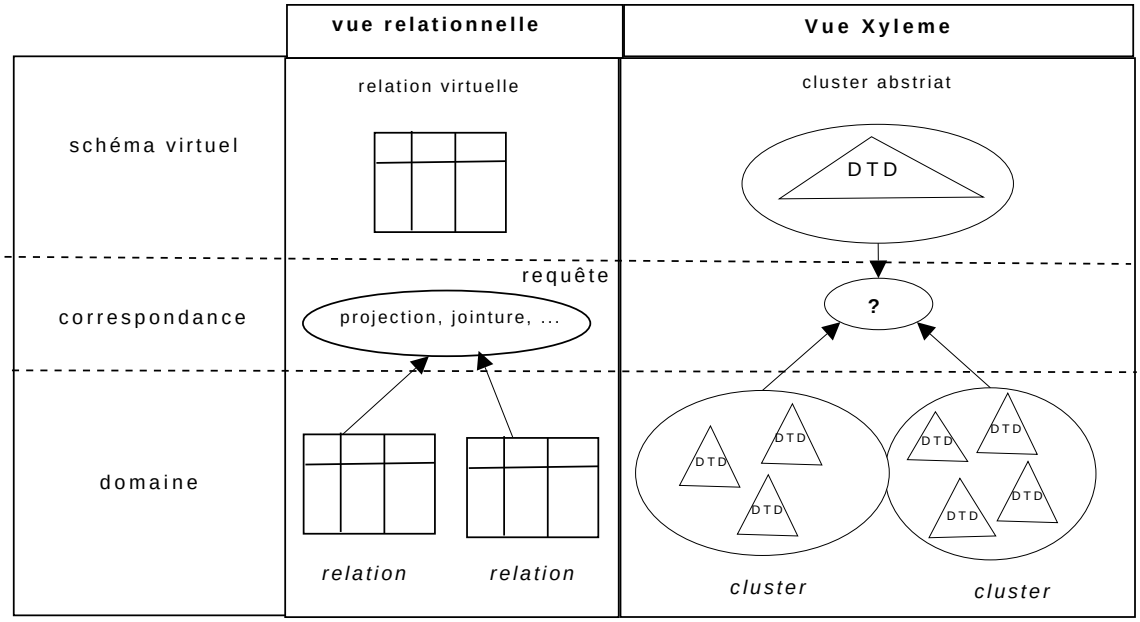


FIG. 3.3 – Vues relationnelles et vues Xyleme.

L'utilisateur doit connaître la structure du document demandé pour pouvoir formuler une requête. Dans le cas d'une base de données XML à grande échelle, où les documents ont des structures différentes, l'utilisateur doit connaître toutes ces structures pour formuler la requête. Dans la suite, on explique comment les vues peuvent être utilisées pour permettre aux utilisateurs de formuler leurs requêtes sans se soucier de connaître la structure de tous les documents en question.

L'idée de [Cluet et al., 2001] est de créer un document abstrait (une vue) qui permet aux utilisateurs de naviguer (chercher) dans un seul document et compter sur le système pour traduire la requête en une requête sur des documents concrets (qui sont inclus dans les bases de données). On suppose que les documents sont structurés, par exemple des documents XML, et stockés dans des entrepôts de données. Les documents peuvent être représentés sous forme d'arborescence comme dans XML.

La définition d'une vue [Cluet et al., 2001] nécessite la mise en place de mécanismes permettant de déterminer comment les requêtes sur des vues sont traduites dans des requêtes sur les données réelles. Ce mécanisme est basé sur la correspondance (*mapping*) de type *path-to-path*, i.e., une vue spécifie la correspondance (*mapping*) entre les chemins (*paths*) dans les structures d'arbres abstraits et d'arbres de documents dits concrets car disposant d'une extension.

Il existe différentes méthodes pour établir la correspondance entre un schéma global abstrait et un schéma concret : la correspondance *tag-to-tag* [Madhavan et al., 2001], [Reynaud et al., 2001], la correspondance *path-to-path* et la correspondance *DTD-to-DTD*. La majorité des projets actuels utilise la seconde catégorie. [Aguilera et al., 2002] étudie les avantages et les inconvénients de chacune de ces approches. Les critères utilisés pour évaluer les approches sont :

– la taille des vues de la correspondance.

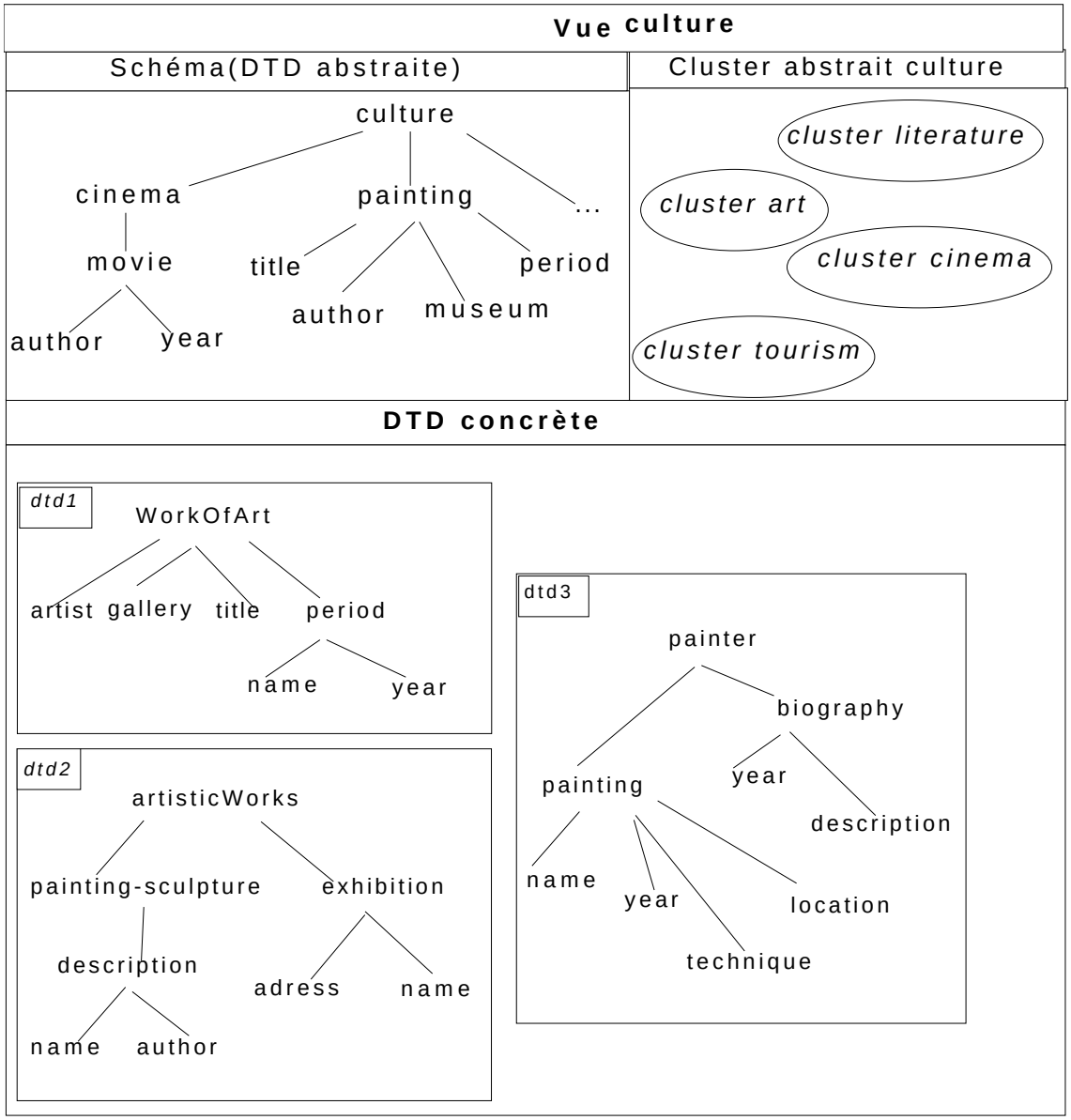


FIG. 3.4 – Exemple de vue sur le domaine culture

– la possibilité de génération automatique de la correspondance.
– la précision des résultats de requêtes.
– le coût de la traduction.

Le résultat de cette étude montre que la correspondance de type *path-to-path* est la meilleure, car elle prend moins de place, elle peut être générée automatiquement et l'exécution de requêtes est plus rapide qu'avec la méthode *DTD-to-DTD*. La table 3.1 résume les qualités et les défauts des différentes méthodes :

| Critère | Stockage | Précision | Rapidité d'exécution | Génération automatique |
|---|---|---|---|---|
| tag to tag | très bien | mauvaise | mauvaise | très bien |
| DTD to DTD | mauvaise | très bien | très bien | mauvaise |
| Path to Path | moyen | moyen | très bien | moyen |

TAB. 3.1 – Comparaison entre les différentes méthodes de correspondance

Dans le système OSIX, on a utilisé la correspondance "path-to-path". L'idée de cette méthode est de préserver le contexte de chaque nœud, aussi bien du côté concret que du côté abstrait, en utilisant les chemins de ces nœuds depuis la racine de leur arbre, c'est-à-dire avec leur adresse absolue dans l'arbre. Par exemple, le nœud *"name"* appartient au *path* "*WorkOfArt/period/name*" et a une interprétation différente du nœud feuille dans le *path* "*person/student/name*".

Dans Xylème, la requête formulée sur la vue est traduite en utilisant la correspondance (*mapping*). Pendant la phase de traduction, le système combine les chemins (paths) concrets en se concentrant sur ceux qui génèrent un sous-arbre de certains documents concrets dans la base de données. Le résultat est l'union de requêtes concrètes construites à partir des combinaisons des arbres valides. Enfin, dans [Xyleme, 2001], les requêtes concrètes sont évaluées en utilisant le language *XQuery* comme language d'interrogation.

Le mécanisme de stockage de vue dans Xylème est ilustré comme suit :

**La distribution des vues dans Xyleme**

Une requête portant sur une vue est compilée traditionnellement en remplaçant le nom de la vue par sa définition au sein de la requête initiale [Aguilera et al., 2002]. Cette technique, qui implique une connaissance complète de la vue lors de la phase de compilation, est difficilement adaptée à Xyleme car elle impose le stockage et la réplication de l'ensemble des vues sur chaque machine interface. Ce problème a obligé les concepteurs à trouver un moyen pour distribuer les vues sur les diverses machines sans avoir un effet négatif sur le processus de traduction de requête. La solution proposée dans Xyleme est de distribuer la vue parmi les machines d'interface et d'index : sur chaque machine d'interface est stocké un arbre qui présente la DTD abstraite de cette vue. Chaque nœud est annoté par un ensemble de *clusters*, où un *cluster* fait partie de l'annotation s'il existe une correspondance de chemin entre la DTD abstraite de la vue et une DTD concrète de ce *Cluster*. Chaque machine index contient les informations de vues relatives aux *clusters*. Elle indexe ces informations sous la forme d'une table des chemins concrets et d'un arbre annoté.

Considérons la vue **culture** de la figure 3.4. En utilisant le principe de distribution, on obtient sur chaque machine interface l'arbre montré dans la partie supérieure de la figure 3.5. Par ailleurs, chaque machine index contient la table et l'arbre annoté dans la partie inférieure de cette figure.

On remarque que la table code de façon compacte la forêt des chemins concrets correspondant au chemin abstrait de la vue. Chaque nœud de cette forêt y est représenté par un identificateur, une balise et l'identifiant de son nœud père (-1 pour un nœud racine). Par exemple, le chemin

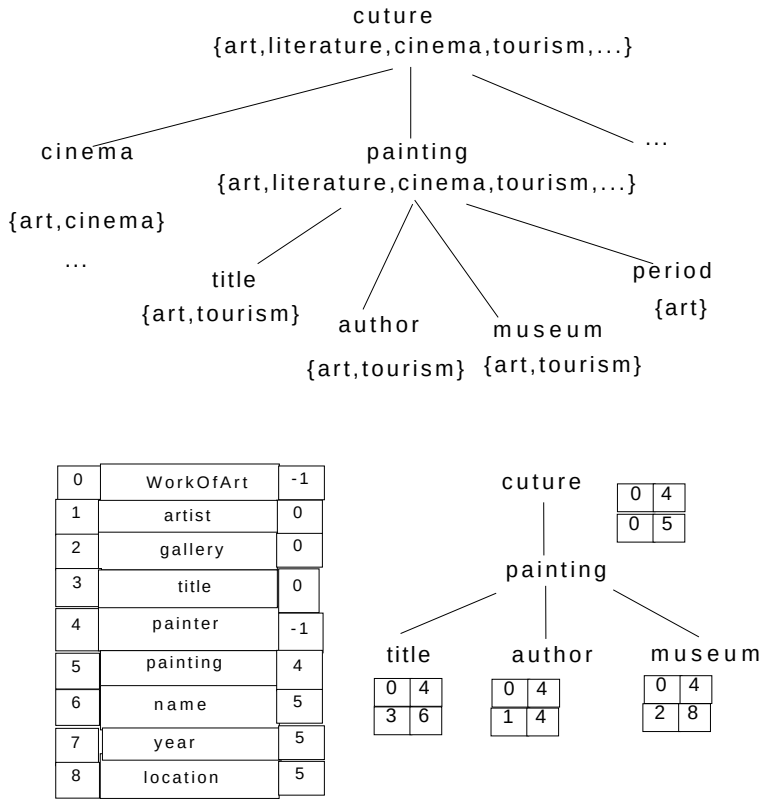


FIG. 3.5 – Distribution de la vue culture

qui a un identificateur **6** est le chemin concret ***painter/painting/name***. L'arbre quant à lui code les correspondances de chemins. Les chemins concrets sont codés dans l'arbre par un couple d'entiers (*root,path*) qui correspondent respectivement à l'identificateur du nœud racine de cette DTD concrète et à l'identificateur du chemin concret. En suivant par exemple le chemin ***culture/painting*** on trouve les deux couples (0,0) et (4,5). Puis, en utilisant la table des chemins concrets on retrouve bien les deux premières correspondances de chemins : (***culture/Painting :Work of Art*** *et* ***culture/painting :painter/painting***)

En résumé, dans le système Xylème, deux phases sont cruciales pour définir les vues sur les sources :
dans la **première phase**, le système crée des DTDs abstraites en classifiant les DTDs concrètes des documents de l'entrepôt. Cette classification utilise une analyse statistique, qui calcule les similarités entre les mots trouvés dans les documents et leurs DTDs.
**La deuxième phase** consiste à établir les connexions sémantiques permettant d'associer les éléments des DTDs abstraites avec ceux des DTDs concrètes. Les techniques de vues présentées dans ce travail peuvent être utilisées dans toutes les applications largement basées sur les échanges de données et l'intégration de données, où les sources données sont très volumineuses et les données sont hétérogènes dans les contenus et les structures. Xyleme a répondu aux autres problèmes que soulève l'intégration sémantique de données. Ainsi, ce système offre des services de haut niveau sur de gros volumes de documents XML. Parmi ces services on peut citer :

- la définition de techniques de stockage efficace et de représentation physique des données ;
- l'interrogation des données et l'optimisation de requêtes en utilisant l'indexation de docu-

ments XML ;
- la gestion des modifications et des mises à jour des données ;
- la maintenance et l'acquisition de données XML du Web.

### 3.4.2 le modèle *XyView*

*XyView* est un modèle de vues pour le développement d'applications utilisateur (formulaire web) ou orientées machine (service web) sur un entrepôt de documents XML hétérogènes et éventuellement sans schéma. Ce modèle de vues permet de visualiser un tel entrepôt comme un tableau de valeurs qui peut être interrogé par des requêtes simples de type sélection/projection ou en utilisant une interface comme l'interface *QBE* [Sebi, 2007].
Les relations dans XyView sont caractérisées par :
- l'entrée n'est pas un schéma relationnel mais un nombre potentiellement élevé de "résumés de données XML" ;
- la vue n'est pas définie explicitement par une requête unique mais implicitement par divers *mappings*, afin d'éviter des doublons et des pertes de données générées par des jointures.

*XyView* a été implémenté comme un ensemble d'outils au-dessus de l'entrepôt XML *Xyleme* [Xyleme, 2001]. Il peut facilement être adapté à n'importe quel système implémentant *XQuery*. Pour faire face à la complexité des données entrées, *XyView* définit les vues en deux étapes :

- La **première étape** traite l'hétérogénéité des données et définit des *mappings* entre les documents hétérogènes (mais sémantiquement liés) et le schéma global (cible). Au moment de l'exécution, cette étape génère des unions.

- La **deuxième étape** correspond à la définition de la vue standard où les données sont agrégées. À l'exécution, cette étape conduit aux jointures.

Comme dans l'architecture générale d'un *wrapper-mediator*, le modèle *XyView* ajoute un niveau intermédiaire qui construit la vue en séparant les unions des jointures et offre un type XML homogène pour les éléments d'une relation universelle.

Les objectifs visées par XyView sont :

1. trouver quels sont les documents nécessaires parmi les divers types de documents dans le système, et quelles sont leurs structures sous-jacentes.
2. comprendre quels sont les éléments XML (et leurs chemins d'accès) impliqués dans chaque requête, et la manière de les combiner pour produire le résultat.
3. les requêtes de l'application doivent être correctement exprimées en XQuery.

L'objectif de XyView est d'optimiser la productivité de nombreuses requêtes pour répondre aux besoins des clients en leur permettant de visualiser la base de données comme quelque chose d'aussi simple que de la formulation d'une requête comprenant des champs qui peuvent être utilisés pour extraire ou filtrer des données.

*XyView* diffère de tout mécanisme standard de vues basé sur la composition des requêtes. La vue dans le modèle XyView n'est pas définie par une requête et n'est pas équivalente à une requête. Plus précisément, une vue XyView [Vodislav et al., 2005] est définie à trois niveaux, par un ensemble de *mappings* et de conditions de jointures, spécifiant comment une requête simple de sélection-projection faite par un utilisateur peut être traduite en XQuery :

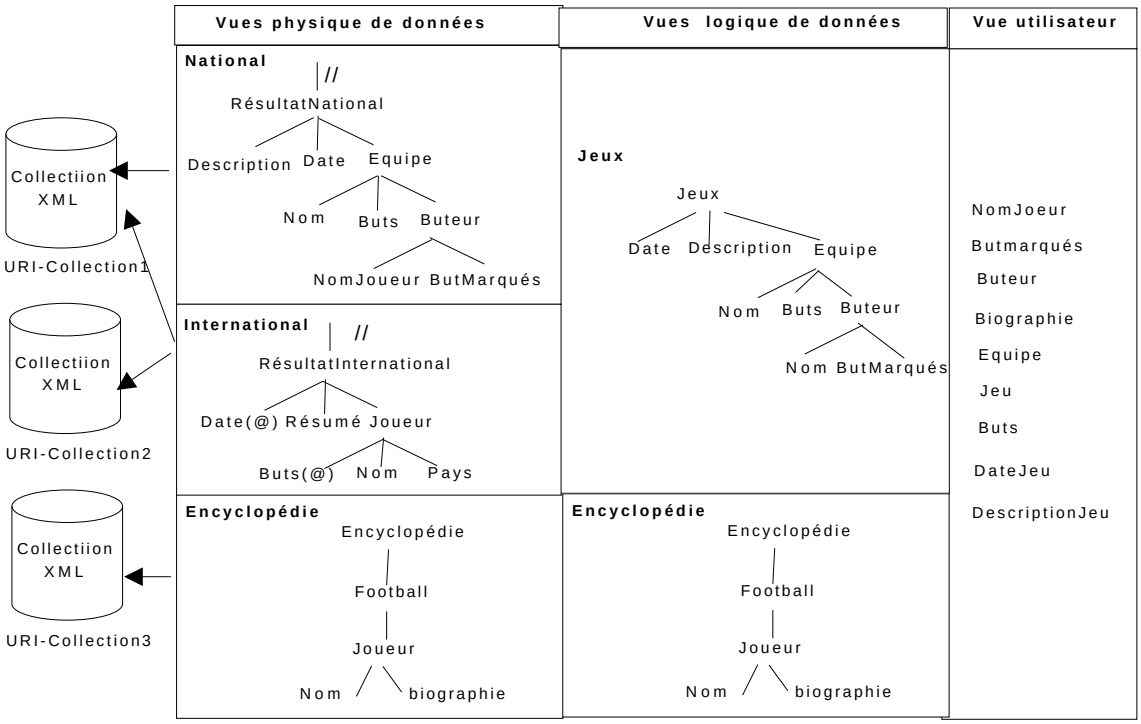


FIG. 3.6 – Définition de la vue à trois niveaux

- Le premier niveau génère des vues physiques (*PDV : Physical Document View*) sur les résumés de données afin de résumer les chemins d'accès XML (*XML access path*) aux informations utiles dans les documents. Elles représentent les données comme elles sont stockées dans l'entrepôt.

- Le deuxième niveau définit des vues logique (LDV : Logical Document View) intégrées sur des vues physiques (PDV). Une vue logique traite l'hétérogénéité des données, en groupant les PDVs ayant une même sémantique dans une même structure d'arbre.

- Le troisième niveau définit la vue utilisateur comme des jointures entre les vues logiques (LDV).

La figure 3.6 illustre la définition de ces trois niveaux :
L'étude du mapping dans *XyView* se fait selon les deux axes suivants :

1. Mapping PDV-LDV.
2. Mapping LDV-vue utilisateur.

Les *mappings* entre les *LDVs* et *PDVs* sont basés sur les correspondances entre les nœuds des arbres *LDV* et *PDV* (voir figure 3.7). Si l'on considère qu'un nœud dans un arbre peut facilement être identifié par son chemin à partir de la racine, on peut noter que cette approche pour représenter la correspondance entre les arbres est proche du mapping *path-to-path*. Bien qu'un nœud dans LDV puisse correspondre à plusieurs nœuds dans PDV, on impose la restriction suivante : un nœud dans une LDV peut être mis en correspondance avec au plus un nœud dans la même *PDV*. Cette restriction élimine toute ambiguïté dans le passage de *LDV* à *PDV* pour le traitement des requêtes. La conséquence est que toute interrogation sur la LDV est traduite

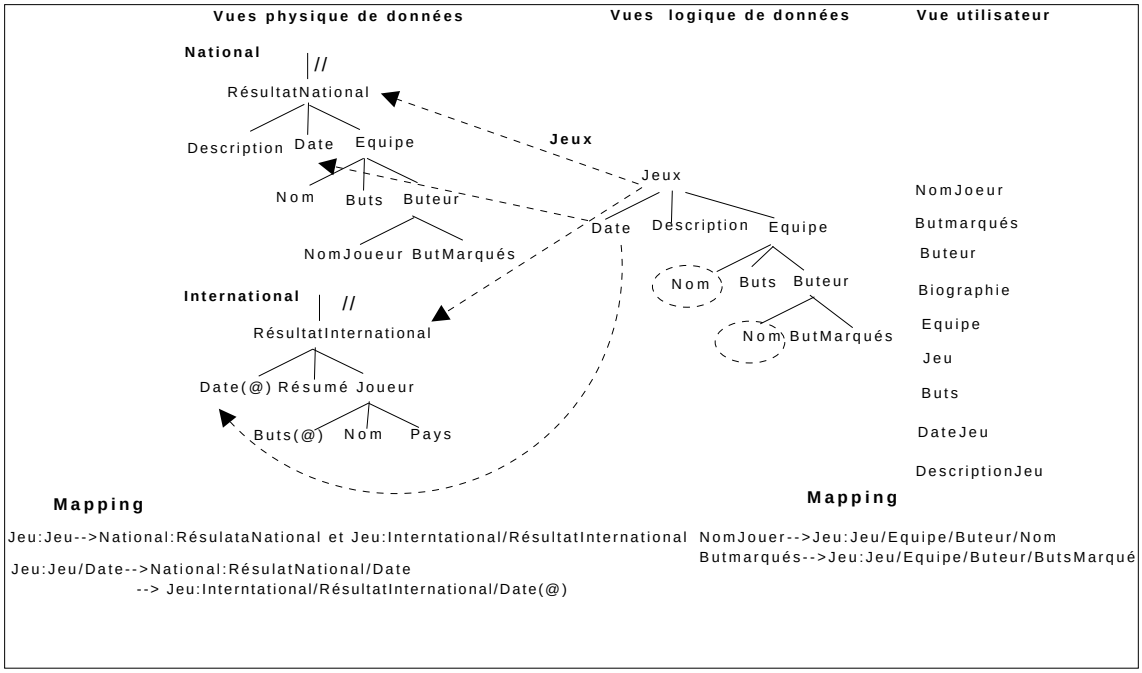


FIG. 3.7 – mapping entre les vues

en une interrogation sur la PDV de manière unique (et facile à calculer).

Une requête sur une vue logique des données est traduite simplement en une union des requêtes sur ses vues physiques correspondantes. Chaque membre de l'union est obtenu par la transformation de chemins de la requête *LDV* en les chemins correspondants dans chaque *PDV*.

la correspondance entre les nœuds dans la vue logique de données et la vue utilisateur est un ensemble de prédicats qui sont utilisés pour joindre les vues logiques des données. Ceci est illustré dans la figure 3.7. Chaque concept a au moins un nœud correspondant dans une *LDV*.

### Traduction d'une requête dans XyView

Dans *XyView* une requête sur une vue logique des données est traduite en un union des requêtes sur les vues physiques correspondantes. L'algorithme de traduction d'une requête **Q** de l'utilisateur est composée de cinq étapes [Vodislav et al., 2005] :

1. Identifier les *LDVs* et les jointures impliquées dans la requête **Q** de l'utilisateur ;
2. Produire un arbre de représentation de **Q** en ajoutant les annotations de la requête aux arbres *LDV* ;
3. Trouver, pour chaque *LDV* annotée, les sous-arbres *PDV* qui correspondent à **Q** ;
4. Générer toutes les combinaisons des jointures entres les *PDV* ;
5. Générer la requête finale XQuery pour faire l'union des combinaisons de l'étape 4.

En résumé, d'après [Sebi, 2007], l'approche *XyView* adapte le paradigme de la relation universelle afin de simplifier à la fois la formulation d'une requête qui est un besoin pour l'utilisateur final et le développement de l'application. Poser des requêtes sur les vues de *XyView* est très simple ; la précision et la pertinence des résultats sont garanties. Le prix à payer est un effort du concepteur de la vue pour créer et maintenir la vue. Toutefois le modèle *XyView* n'est pas basé sur

la requête, il ressemble plutôt aux systèmes d'intégration des données XML basés sur un médiateur comme [Baru et al., 1999], [Ives et al., 2002], [Cluet et al., 2001] et [Fundulaki et al., 2002]. Les vues dans XyView sont définies à travers un ensemble de *mapping path-to-path*. Ceci évite tous les problèmes de jointure inutiles, la perte d'information et les doublons. Par ailleurs, cela permet l'utilisation d'outils graphiques, ce qui simplifie la tâche du concepteur de vues ainsi que celle de la maintenance.

### 3.4.3 le système *VIMIX* (*View Model for Integration of XML sources*)

Ce modèle permet l'intégration de sources des données XML hétérogènes par la définition des vues sur ces différentes sources. Le langage de définition de vues utilisé est inspiré de *XML-QL* où on utilise le *pattern-matching*, qui spécifie les données XML à extraire dans les différentes sources et qui spécifie le résultat d'une requête en décrivant un arbre XML. Afin de restructurer les données extraites des sources, *VIMIX* propose des fonctions pour faire l'union et la jointure de ces données réparties dans les sources [Baril, 2003].

VIMIX représente les données XML en utilisant un graphe de données où on distingue trois types de nœuds : élément, attribut et texte, et deux types de liens : les liens de composition, qui représentent l'imbrication des éléments dans les données, et les liens de référence entre les éléments que l'on définit en utilisant les attributs ID/IDREF(S) pour le partage des éléments.

VIMIX réalise l'intégration de données en passant par trois étapes :

1. explicitation des données que l'on veut extraire de différentes sources. Pendant cette étape, on spécifie des motifs sur les sources (*source-pattern*). Ces motifs représentent les feuilles du graphe des correspondances (*mappings*).

2. définition des fragments et des jointures permettant de construire le graphe des correspondances.

3. Spécification de vues qui représentent le schéma médiateur de *VIMIX* : à partir des nœuds du graphe des correspondances on restructure les données extraites.

La figure 3.8 résume le mécanisme d'intégration de données proposé par VIMIX .

Pour comprendre les différentes étapes du processeur de l'intégration avec VIMIX, [Baril, 2003] utilise un exemple d'une source de données bibliographiques validée par la DTD suivante :

```
<?xml version="1.0" encoding="ISO8859_1" ?>

<!-- element racine -->
<!ELEMENT bibliographie (auteur+, publication*) >

<!-- auteurs -->
<!ELEMENT auteur (nom, prenom, email?, web?) >
<!ATTLIST auteur
          id ID #REQUIRED>
```

[4] Un fragment est composé d'une liste de sources de données dont on peut faire l'union.

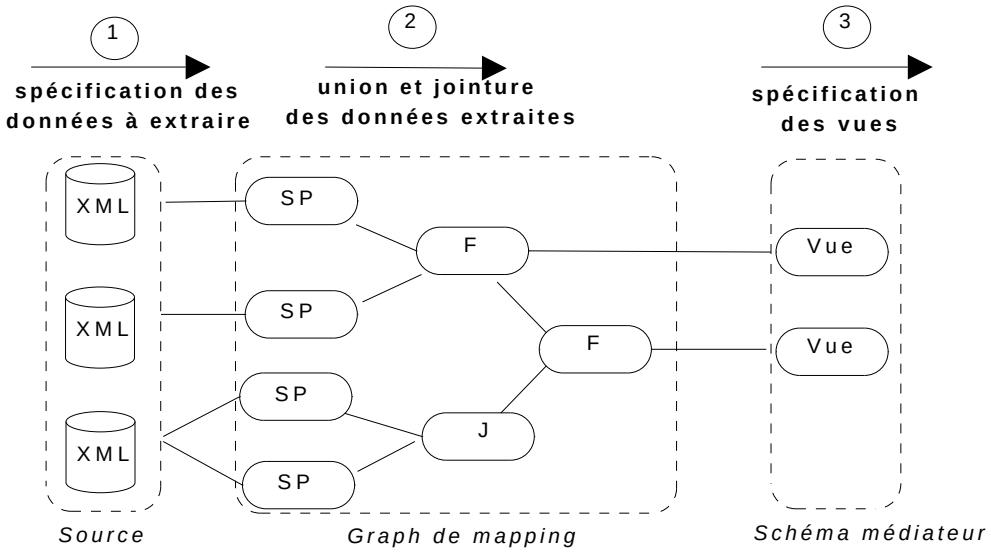


FIG. 3.8 – Intégration de données avec VIMIV.

```
    <!-- publications -->
    <!ELEMENT publication (titre, type, annee, titre-livre, pages, isbn?) >
    <!ATTLIST publication
              id ID #REQUIRED
              auteurs IDREFS #REQUIRED
    >

   <!-- PCDATA -->
   <!ELEMENT nom (#PCDATA) >
   <!ELEMENT prenom (#PCDATA) >
   <!ELEMENT email (#PCDATA) >
   <!ELEMENT web (#PCDATA) >
\end{quotation}
  <!ELEMENT titre (#PCDATA) >
  <!ELEMENT type (#PCDATA) >
  <!ELEMENT annee (#PCDATA) >
  <!ELEMENT titre-livre (#PCDATA) >
  <!ELEMENT pages (#PCDATA) >
  <!ELEMENT isbn (#PCDATA) >
```

L'objectif est d'extraire les noms et les prénoms des auteurs dans les différents sources.

### Processus d'intégration de données

**Etape 1 : spécification des données à extraire**

Cette étape se fait *par pattern-matching* : les données des sources sont liées à des variables qui sont déclarées dans un motif (*pattern*) décrivant la source. La définition des variables se fait en utilisant un mécanisme d'**axes de recherche**. Ce mécanisme est similaire à XPath [5] pour

[5] *Xml Path Language* (xpath) 1.0. W3C Recommendation, *http ://www.w3.org/TR/1999/REC-xpath-19991116.*

localiser les parties d'un document. Le motif (*pattern*) suivant est appelé *"ap-auteur-biblio"* ; il récupère les noms et les prénoms des auteurs :

```
<source-pattern name="sp_auteurs_biblio" source="biblio">
   <search-axis function="children">
    <source-node reg-expression="auteur" type="element">
      <search-axis function="children">
         <source-node reg-expression="nom"
                      type="element"
                      bindto="nom">
         </source-node>
         <source-node reg-expression="prenom"
                     type="element"
                     bindto="prenom">
       </source-node>
      </search-axis>
  </source-node>
 </search-axis>
</source-pattern>
```

Dans*VIMIX*, les données extraites par les *patterns* sont stockées dans des tables relationnelles. Par conséquent, il est possible de faire la restructuration de données en utilisant l'algèbre relationnelle. *VIMIX* adapte deux opérations de cette algèbre qui sont pertinentes pour l'intégration des données : l'union et la jointure. L'opération d'union prend comme entrée plusieurs opérandes, qui sont des résultats des patterns, des unions ou des jointures. Le résultat est stocké dans une table relationnelle dont les attributs sont calculés par l'union des attributs des tables opérandes. Les valeurs de cette table sont construites par l'union de n-uplets des sources en utilisant la valeur *NULL* pour les attributs qui n'existent pas dans une source. Contrairement à l'union dans l'algèbre relationnelle, l'union dans *VIMIX* peut être appliquée lorsque les sources ont des schémas différents.

Revenons à l'exemple et considérons une deuxième source de données, **biblio-Lirmm**, qui contient les publications et les auteurs du laboratoire. Soit **sp-auteurs-lirmm** un motif qui permet d'extraire les auteurs de cette source ; il contient les information suivantes sur chaque auteur : nom, prénom et email.

**Etape 2 : définir l'union des données**

On définit l'union des données provenant de différentes sources en utilisant des fragments : un fragment est une unité sémantique permettant d'intégrer des données provenant de sources multiples et hétérogènes. Le fragment suivant spécifie l'union des auteurs dans deux motifs différents :

```
<fragment name="f_auteurs" date="sp_auteurs_biblio_lirmm  sp_auteurs_biblio">
  <restrictions>
    <restrictions  variable="nom"  priority="sp_auteurs_biblio_lirmm">
  </restrictions>
</fragment>
```

La deuxième étape de l'intégration de données peut aussi définir des jointures des données provenant de différentes sources.

**Etape 3 : la spécification d'une vue**

Le n-uplet d'une vue possède les propriétés suivantes :

- un nom, pour identifier la vue,
- une source (motif, fragment ou jointure), qui contient les données de la vue,
- un motif, qui décrit la structure de la vue,
- l'ordre des données,
- les niveaux de regroupement des données.

Considérons le fragment `f-auteurs`, qui spécifie l'union de deux sources de données bibliographique pour extraire les auteurs. L'exemple suivant spécifie une vue qui présente les noms et les prénoms des auteurs de manière uniforme :

```
<view name="v_auteurs" source="f_auteurs" group-by="" order-by="">
  <result-node type="element" value="auteur">
  <result-node type="attribute" value="nom">
  <result-node type="expression" value="text(nom)" />
  </result-node>
  <result-node type="attribute" value="prenom">
  <result-node type="expression" value="text(prenom)" />
  </result-node>
  </result-node>
</view>
```

Le résultat de la vue sera structuré comme suit : chaque élément "auteur" possédera deux attributs contenant le "nom" et le "prénom".

### Stockage de vues dans *VIMIX*

Dans le système *VIMIX* les vues sont matérialisées dans un *S*GBD relationnel. Cette matérialisation permet de construire un entrepôt de données XML à partir d'un système d'intégration défini avec *VIMIX*. Le schéma médiateur de cet entrepôt est défini à partir des schémas des vues.

L'architecture de stockage dans *VIMIX* évite les redondances des données XML dans l'entrepôt parce qu'elle sépare le stockage des données de celui des méta-données (les *mappings*). Cette architecture est basée sur un schéma générique pour stocker les données XML des sources. Les méta-données (*mappings*) qui spécifient les données extraites par des motifs sur les sources, des fragments et des jointures, sont stockées dans des tables.

### Stockage des données

Le schéma générique utilisé par *VIMIX* pour stocker les données est décrit dans la figure 3.9 ; il est destiné à stocker des nœuds qui viennent des sources, sans stocker toutes les données des sources. Pour chaque source (ou document) on a besoin simplement de connaître son identifiant et son URL. La table **Document** contient les URLS des sources de données. Les table **Element** et **Attribut** sont des dictionnaires des éléments et des attributs stockés dans l'entrepôt. La table **XmlNode** permet de stocker les nœuds des sources. Les tables **Children** et **Descendants** contiennent respectivement les liens de référence et de descendance entre les éléments de l'entrepôt.

| Table | Colonnes |
|---|---|
| Document | doc ID, url |
| Element | elem ID, name |
| Attributt | att ID, name |
| XmlNode | node ID, elem ID, att ID, value, doc ID |
| Children | fatherID, childID, rank |
| Descendants | fatherID, childID, rank |

FIG. 3.9 – Schéma générique pour le stockage de données XML.

**Stockage des *mappings***

L'idée principale de *VIMIX* est que la matérialisation des parties des vues est meilleure que la matérialisation complète de la vue, afin de permettre l'amélioration de la maintenance incrémentale et la réduction de l'espace de stockage. Les tables stockant les *mappings* peuvent être organisées en un graphe de *mappings*, où chaque table est un nœud du graphe. On distingue trois types de nœuds permettant de représenter les différentes tables contenant les mappings :

1. Les nœuds de type **source** représentent les tables contenant les données extraites par des motifs. Ces nœuds sont les feuilles du graphe.
2. Les nœuds de type **fragment** représentent les tables contenant les données intégrées dans les fragments. Ces nœuds ont pour fils les nœuds représentant les sources utilisées pour définir le fragment.
3. Les nœuds de type **join** représentent les tables contenant les données intégrées dans une jointure. Ces nœuds ont pour fils les nœuds représentant les sources utilisées pour définir la jointure.

En résumé, *VIMIX* présente une méthode de stockage et des algorithmes pour la maintenance des vues XML stockées dans un **RDBMS**. Sa méthode de stockage sépare le stockage de données XML de celui des méta-données qui décrivent les *mappings*. L'identificateur utilisé pour stocker la définition d'une vue permet la maintenance incrémentale, dans le sens où la matérialisation de la vue ne nécessite pas de recalculer toutes les données stockées pour maintenir les vues XML. Une autre avantage de cette méthode est lié à l'utilisation d'un graphe avec des vues multiples, qui permet une représentation des sous-ensembles d'expressions communes à différentes vues. En conséquence, cela permet de réduire l'espace de stockage et le temps de maintenance d'une vue.

## 3.5 Conclusion

La notion de vue est essentielle dans le domaine de l'intégration des données. Elle permet à des utilisateurs divers de voir les données selon des points de vue différents. Elle permet aussi d'utiliser une structure unique pour accéder aux données provenant de sources hétérogènes. Les vues peuvent être utilisées pour construire le schéma global d'un système d'intégration de données

sur lequel l'utilisateur peut poser ses requêtes. Les vues peuvent être virtuelles (dans le cas d'un système médiateur) ou matérialisées (dans le cas d'un entrepôt de données).

Dans ce chapître, nous avons présenté la définition des vues dans les différents modèles de données : le modèle relationnel, le modèle orienté-objet et le modèle semi-structuré. Nous nous sommes concentrés sur le modèle de données semi-structuré, et plus précisément les données XML. On a présenté un état de l'art sur les modèles de vues existants pour l'intégration et l'interrogation de sources de données XML. Cette étude nous a aidés à préciser les différents étapes de notre approche (OSIX : Osiris-based System for the Integration of XML documents) conçu pour l'intégration des documents XML. Cette approche est basé sur le système Osiris, qui partage des données au travers de vues. Dans le chapître 5, nous présentons une comparaison entre notre modèle et les autres modèles étudiés dans ce chapitre : Xyleme, XyView et VIMIX.

Avant de décrire notre approche, on commence le chapitre suivant par une description détaillée du système OSIRIS et ses concepts de base.

# Chapitre 4

# Le système Osiris

Ce chapitre est une présentation du système Osiris. Dans la première partie, la description du modèle Osiris permet d'introduire la notion de P-type. La seconde partie présente le principe de classement des objets qui est utilisé dans Osiris et introduit les notions de sous-domaine stable et d'Eq-classe. Ce chapitre se termine par la présentation de l'indexation d'objets en Osiris et l'utilisation de cette indexation pour réaliser l'optimisation sémantique de requêtes.

## 4.1 Introduction générale

Le système Osiris est un système de représentation et de gestion des données et des connaissances (SGBD-BC) qui implante le modèle de données des P-types. Le concept de P-type a été créé en 1984 par A. Simonet [Simonet, 1984]. Il a deux objectifs principaux qui sont le partage des données au travers de vues (P-type = Type Partagé) et la vérification automatique de contraintes d'intégrité élaborées.

Cette notion de vue a été délaissée dans la première génération de SGBD orientés objet au motif qu'elle pouvait être simulée par la notion d'héritage. Sans sous-estimer l'intérêt évident de l'héritage dans le modèle objet, on constate, cependant, que cette approche induit un grand nombre de classes supplémentaires qui polluent le modèle. L'exemple ci-dessous (voir 4.1) en est une illustration. Il modélise le concept de Personne et exprime qu'une personne peut être étudiant et/ou enseignant et/ou sportif.

Cette multiplication de classes est due au fait qu'un objet ne peut être instancié que dans une classe et une seule. Il apparaît évident qu'avec un nombre important de classes, cette approche n'est plus viable. Cette même remarque a été faite par E. Bertino dans [Bertino & Guerrini, 1995] où elle propose un modèle dans lequel les objets peuvent être instanciés dans plusieurs classes.

De plus, dans un contexte de bases de données, les objets évoluent au cours de leur existence. A un moment donné, une personne cessera d'être étudiant. Il faudra alors la supprimer de la classe dans laquelle elle était instanciée et la récréer dans une autre classe. Dans la plupart des SGBD orientés objet, cette évolution se traduit par un changement de l'OID de l'objet, ce qui a pour conséquence d'invalider tous les liens que cet objet pouvait avoir avec d'autres objets. Le P-type est la solution proposée dans le modèle Osiris pour répondre à ces questions.

## 4.2 Description d'Osiris

Le système Osiris implémente le modèle des P-types. Ce modèle a été initialement conçu dans l'optique d'un SGBD privilégiant le partage d'information à travers des points de vue ou

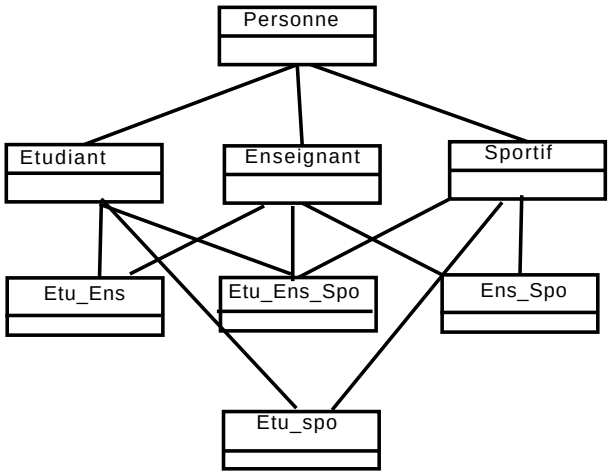


FIG. 4.1 – Classes nécessaires pour exprimer la multi-appartenance par héritage

vues : les objets d'un domaine d'étude sont groupés en familles sémantiques, ou P-types, et à l'intérieur d'un même P-type, un objet peut être perçu selon différentes *vues*. Une vue est définie dans un P-type et un seul, et est caractérisée par des *propriétés* logiques, exprimées au travers des attributs [6] et des contraintes (appelées aussi assertions ou axiomes) dans un contexte de spécialisation stricte, qui peut être simple ou multiple. Ces propriétés constituent les conditions nécessaires (cas de la vue minimale) ou des conditions nécessaires et suffisantes (pour les autres vues) d'appartenance d'un objet à une vue.

Lors d'une affectation impérative d'un objet à une vue, les propriétés de la vue jouent le rôle de contraintes d'intégrité à vérifier. La méthodologie utilisée pour la vérification de ces contraintes permet également de déterminer toutes les vues valides de l'objet, c'est-à-dire les vues dont l'objet satisfait les propriétés logiques, ce qui correspond au classement d'objet (*instance classification*) dans les bases de connaissances. Cette méthodologie détermine aussi les vues dont l'objet contredit les propriétés, ce sont ses vues *invalides*. Enfin, toutes les autres vues du P-type constituent les vues *potentielles* de l'objet, ce qui peut arriver lorsque certaines valeurs d'attributs sont inconnues.

Les objets d'un P-type sont susceptibles d'être modifiables. Cette modification peut être la conséquence de la modification de valeurs d'attributs (cas des bases de données) ou plus simplement la conséquence de la découverte de la valeur d'un attribut ayant jusqu' alors une valeur inconnue (cas des données ou des bases de connaissances). A chaque mise à jour d'un objet, il est automatiquement reclassé, ce qui permet de déterminer les vues valides, invalides et potentielles dans son nouveau état.

La méthodologie utilisée pour le classement s'appuie sur une partition de l'espace des objets, appelée *Espace de Classement*, obtenue par une analyse statique de l'ensemble des propriétés logiques définies dans toutes les vues d'un P-type. Cette partition est également utilisée pour *indexer* les objets de la base et conduit naturellement à une *optimisation sémantique des requêtes* sans que, pour optimiser les requêtes, il soit nécessaire de classer explicitement les vues entre elles évitant ainsi un processus de complexité NP tout en bénéficiant d'avantages comparables [Simonet, 1996].

[6] Un attribut peut être vu comme une contrainte existentielle.

## 4.3 Concepts de base

### 4.3.1 Concept d'objet

La notion d'objet du monde réel partagé et susceptible d'être perçu selon plusieurs points de vue par différentes catégories d'utilisateurs sous-tend la notion d'*identité* d'une instance, ou objet informatique. Ce concept est donc un des concepts susceptibles d'être définis dans la vue minimale d'un P-type. Il doit être défini si l'application nécessite la manipulation d'objets partagés par plusieurs catégories d'utilisateurs.

Un objet [7] appartient à un P-type et un seul, et satisfait certaines de ses vues, dont la vue minimale. Lors des mises à jour de l'objet, celui-ci est reclassé : dans le nouveau état, ses vues peuvent être différentes de celles de l'état antérieur. Dans tous les cas, l'objet appartient toujours à la vue minimale de son P-type.

Dans la suite, nous présentons les principales notions du modèle P-types. Une formalisation du concept de P-type est ensuite proposée. Enfin, les principales fonctionnalités du système Osiris, système qui implémente le modèle des P-types, sont présentées.

### 4.3.2 Concept de P-type

Le modèle des P-types (**p** pour **p**artagé), a été initialement conçu pour les bases de données [Simonet, 1984] mais s'avère également adapté aux besoins des bases de connaissances, grâce en particulier à sa fonction de classement d'objet. Par rapport aux modèles de représentation de données et des connaissances, la caractéristique innovante de P-types est que la *vue*, qui définit un *point de vue* sur une famille d'objets, en constitue le concept central : on ne définit pas un P-type puis ses vues, mais on définit un P-type *par* ses vues. Un objet est une instance d'un et un seul P-type, mais il peut appartenir à plusieurs de ses vues et en changer pendant sa durée de vie. Le classement d'un objet dans les vues de son P-type est une caractéristique inhérente au modèle. C'est pourquoi Osiris, système qui implémente le modèle des P-types, peut offrir des fonctionnalités pour l'aide à la décision, les alertes, l'optimisation sémantique de requêtes, etc. Dans nos travaux nous avons approfondi l'utilisation du concept de P-type dans l'intégration de sources hétérogènes de données.

Un P-type permet de définir une collection d'objets du monde réel ayant une même sémantique et susceptibles d'être perçus selon plusieurs points de vue, nommés vues, par différentes catégories d'utilisateurs, éventuellement une seule. Parmi les vues d'un P-type, il existe une vue privilégiée, la racine de la hiérarchie de vues. Cette vue, nommée vue minimale, définit le plus petit nombre de propriétés (attributs et contraintes) que tout objet du P-type doit satisfaire pour appartenir au P-type. Elle définit donc la condition nécessaire et suffisante d'appartenance d'un objet à un P-type.
La construction des vues d'un P-type suit une méthodologie descendante : on construit d'abord la vue minimale, puis ses autres vues par enrichissement strict de la vue minimale ou de vues déjà définies. Toutes les vues sont définies par rapport à la vue minimale par héritage simple ou multiple.

### 4.3.3 Spécification d'un P-type

La spécification d'un P-type se fait par la donnée de ses fonctions et de ses contraintes. Contrairement au cas habituel des langages de programmation (Java, C++) ou des modèles

[7] Dorénavant, le terme *objet* désignera un objet informatique. Les termes *objet réel* ou *objet* du *monde réel* étant utilisés explicitement pour désigner les objets du monde réel.

objets des SGBDOO classiques [Cattell, 1993], où l'ensemble des attributs et des contraintes d'une classe est donné de manière plate, la définition d'un P-type se fait par la donnée d'une *hiérarchie de vues*, le type du P-type étant déduit à la compilation.

Osiris distingue plusieurs catégories de fonctions : les *attributs*, les fonctions de mise à jour des *attributs modifiables* et celles associées aux *attributs calculables*, et enfin celles sous-tendues par la *clé externe*. Parmi les contraintes possibles, nous nous intéressons particulièrement aux contraintes d'*existence d'attributs*, aux *contraintes de domaine* et aux *dépendances inter-attributs*. Dans la suite, nous présentons un exemple de P-type, puis nous proposons une formalisation de ce concept. Le langage utilisé est celui du système Osiris, système qui implémente le modèle des P-types.

**Attributs**

En Osiris, un attribut est une fonction unaire dont le domaine est le P-type, et le codomaine est soit un P-type soit un type classique (prédéfini, structure ou collection).

La déclaration de l'attribut $Attr_i$ dans la vue $Vi$ du P-type T est définie par la grammaire :

$attr_i$ : type ;
type : typeClassique | P-type
typeClassique : typePrédéfini | struct ([$attr$ : $type$])$^{1..*}$ | setOf (type)
typePrédéfini : INT | FLOAT | STRING | CHAR | BOOLEAN | DATE

Des exemples d'attributs sont :

age : INT
adresse : struct (num : INT ; rue : STRING , cPostal : INT ; ville : STRING)
adresse : ADRESSE
prénoms : setOf (STRING)
coordonnées : setOf (struct(num : INT ; rue : STRING , cPostal : INT ; ville : STRING))
amis : setOf (struct (ami : PERSONNE ; téléphone : STRING))
proprietaire : SetOf (PERSONNE)
estProprietaire : SetOf (VOITURE)

Afin d'augmenter la déclarativité du langage et de minimiser la tâche du concepteur, tout en garantissant la cohérence de la spécification, un certain nombre de propriétés peuvent être affectées à un attribut au travers de qualifieurs. Parmi ces qualifieurs, nous nous intéressons aux qualifieurs dont nous avons besoin dans nos travaux d'intégration. Ainsi, nous nous intéressons particulièrement à quatre qualifieurs qui permettent d'exprimer qu'un attribut : 1) a un attribut *inverse* ; 2) est susceptible d'avoir une valeur *inconnue* ; 3) est susceptible d'avoir une valeur *indéfinie* ; 4) est un attribut *classificateur*.

***Attribut inverse***

Lorsque le type d'un attribut est un P-type ou un setOf (P-type), l'utilisateur doit déclarer le nom de l'attribut inverse si celui-ci est également significatif pour l'application. Cette déclaration permet au système d'assurer l'intégrité référentielle, c'est-à-dire de garantir que tout objet référencé existe bien dans le système. Elle lui permet aussi de mettre à jour automatiquement l'attribut inverse lors des mises à jour.

```
view VOITURE
  attr propriétaireVoiture : PERSONNE;
  reverse possedeVehicule   -- nom de l'attribut inverse, déclaré dans la vue PERSONNE.
```

En présence d'une telle déclaration, lorsque, par exemple, on affecte à l'attribut *propriétaireVoiture* de la voiture $v_i$ la personne $p_j$ (existante), le système informatique met automatiquement à jour l'attribut *possedeVehicule* de la personne $p_j$, en affectant $v_i$ à $p_j.possedeVehicule$.

***Valeur indéfinie Vs valeur inconnue***

Dans les bases de données, les valeurs indéfinie et inconnue sont en général confondues. Ainsi, en SQL la constante NULL est utilisée indifféremment pour dénoter une valeur inconnue ou indéfinie (au sens « non-existante »), voire même « non obligatoire pour l'application ». En Osiris, la distinction entre la valeur indéfinie et la valeur inconnue est explicite. L'attribut nbGrossesses du P-type PATIENT est un exemple typique d'un attribut dont la valeur peut être indéfinie. En effet, si pour les patients de sexe féminin la valeur de l'attribut nbGrossesses est toujours définie (elle peut avoir la valeur zéro, voire inconnue), pour un patient de sexe masculin elle est toujours indéfinie (de la même manière que la racine carrée d'un nombre négatif est indéfinie).

La distinction indéfini/inconnu est cruciale dans les systèmes chargés de la gestion d'informations critiques, comme les systèmes d'information en santé. Par exemple, pour un patient, savoir que la valeur de son attribut typeDeCancer est indéfinie (et en conséquence, il n'a pas de cancer) ou qu'elle est inconnue (éventuellement, il peut avoir un cancer mais on ne le connaît pas) n'est sans doute pas indifférent.

La constante **null** appartient au domaine de tout attribut auquel le qualifieur **undefined** a été attribué (explicitement ou implicitement).

De plus, tout attribut est par défaut *not mandatory* et peut donc prendre pour valeur la constante **unknown**. Cette constante est donc implicitement affectée à tout attribut d'un objet pour lequel on ne connaît pas la valeur. Elle permet de prendre en compte les valeurs manquantes des objets, ce qui est très utile lors de l'intégration de sources hétérogènes de données qui, outre leurs propres valeurs manquantes, sont susceptibles d'avoir des attributs manquants par rapport à un schéma global d'intégration.

***Attribut classificateur***

Un attribut est dit classificateur si sa valeur doit être utilisée pour le classement des objets d'un P-type. Par défaut, dans un P-type, tout attribut pour lequel il existe au moins une contrainte de domaine ou une DIA (*Dependance Inter-Attributs*) est un attribut classificateur. Les attributs classificateurs sont au centre des processus de classification, d'indexation et d'optimisation sémantique des requêtes.

### Contraintes

La sémantique d'un P-type est exprimée au travers de contraintes, nommées assertions. le système Osiris s'intéressé aux contraintes portant sur le domaine des attributs ; elles se décomposent en trois grandes catégories :

1. Les contraintes de domaine élémentaires,
2. les Dépendances Inter-Attributs (DIAs),
3. les Contraintes sur les Vues (CV).

***Contraintes de domaine***

Les contraintes de domaine élémentaires, appelées par la suite contraintes de domaine, restreignent les domaines des attributs. Lors de la définition d'un P-type, elles peuvent être utilisées dans deux situations :

1. lors de la déclaration d'un attribut propre d'une vue,
2. lors de la spécialisation d'une vue $V_i$ , pour les attributs hérités par une vue fille.

Les contraintes de domaines sont des prédicats élémentaires de la forme $Attr_i \in dom_{ij}$, où $dom_{ij}$ est un sous-ensemble de l'ensemble d'intérêt du type de l'attribut $Attr_i$ .
Des exemples de contraintes de domaine sont :

$age \leq 18$ ;
$sexe = m$ ;
$sMilitaire \in \{effectué, non-effectué, sursitaire\}$ ;
$possedeVehicule \in DIESEL$ – DIESEL est une vue du P-type VEHICULE, et détermine donc un sous-ensemble des objets du P-type VEHICULE.

A noter que les deux premières contraintes peuvent se réécrire en :

$age \in$ [Inf..18]
$sexe \in \{m\}$

***Les Dépendances Inter-Attributs***

Les **D**épendances **I**nter-**A**ttributs (DIAs) sont des clauses de Horn dont les littéraux sont les prédicats élémentaires définis ci-dessus et les connecteurs sont les opérateurs logiques {et, ou, non, $\Rightarrow$ }. Des exemples de DIAs du P-type PERSONNE sont :

$age \in [25, 65]$ – DIA dégénérée
$age \leq 16 \Rightarrow sMilitaire \in \{non-effectué\}$
$age \in [5, 14] \Rightarrow$ possedeVehicule $\in$ (VELO ou VELOMOTEUR)

***Les Contraintes Sur Les Vues***

Les Contraintes sur les Vues (CV) sont des formules propositionnelles $\Phi(Vi)$ sur les vues (d'un même P-type), utilisant les connecteurs **and**, **or** et **not**.
Des exemples de CV sont :

SALARIE
SALARIE **or** BOURSIER
ETUDIANT **and not** SENIOR

Les contraintes sur les vues sont utilisées :

- implicitement lors de la définition d'une vue d'un P-type par la spécialisation d'autres vues déjà existantes. Dans ce cas, le connecteur classique est le connecteur **and**, noté « ,» dans le langage Osiris.
  Par exemple, **view** MONITEUR : ETUDIANT, ENSEIGNANT.
  En Osiris on peut aussi utiliser les connecteurs **or** et **not** lors de la définition des vues d'un P-type.
- Explicitement :
  1. lors de la définition d'une vue pour introduire des contraintes de domaine sur des attributs dont le domaine est un sous-ensemble d'objets d'un P-type.
     Par exemple, amis : setOf (MINEUR) ou encore amis : setOf (MINEUR **or** SPORTIF),
  2. dans la définition de prédicats élémentaires et DIAs,

3. dans la définition de requêtes. Par exemple, (ETUDIANT **and** SPORTIF | Age > 23),
4. dans la définition de vues dynamiques : Q1 : (ETUDIANT **and** SPORTIF| Age > 23).

**Hiérarchie de vues**

La donnée des vues d'un P-type se fait de manière incrémentale et modulaire : on spécifie d'abord sa vue racine, nommée *vue minimale*, puis ses autres *vues* par enrichissements stricts successifs de vues déjà définies. Un enrichissement peut être simple ou multiple. Lors d'un enrichissement, outre la restriction stricte du codomaine des fonctions héritées des vues mères, de nouvelles fonctions ou de nouvelles contraintes peuvent être définies.

***Vue minimale***

La spécification de la vue minimale d'un P-type se fait par la donnée de son nom, de ses attributs et de ses contraintes. Outre les contraintes de domaine et les DIAs, la contrainte d'unicité peut aussi être fournie. Elle permet au système informatique de déterminer si deux instances représentent ou non le même objet réel. Par défaut, un P-type a le même nom que sa vue minimale (Cf. exemple ci-dessous).

***Vue non minimale***

Une vue non minimale est obtenue par un enrichissement d'une ou de plusieurs vues déjà définies d'un P-type. Un enrichissement - ou spécialisation - peut être simple (une seule vue mère) ou multiple (plusieurs vues mères). Lors d'un enrichissement, outre la restriction stricte du codomaine des fonctions héritées des vues mères, de nouvelles fonctions ou de nouvelles contraintes peuvent être définies. Dans l'exemple ci-dessous, ENSEIGNANT est une vue obtenue par spécialisation simple, tandis que MONITEUR est une vue obtenue par spécialisation multiple.

### 4.3.4 Exemple de P-type

Comme nous l'avons déjà dit, par défaut, un P-type a le même nom que sa vue minimale. Lorsqu'il y a une ambiguïté entre le P-type et la vue minimale, nous ajouterons au nom du P-type le symbole *. Par exemple, PERSONNE et PERSONNE* dénotent respectivement la vue minimale et le P-type au niveau du *langage externe* [8], et Personne dénote le type du P-type au niveau interne. Le programme suivant définit le P-type PERSONNE et quelques unes de ses vues.

**view** PERSONNE — - Vue minimale du P-type PERSONNE
**attr**
id : INT ;
nom : STRING ;
pere : PERSONNE ;
sexe : CHAR **in** f, m ; — - assertion a1
dateNaissance : DATE ;
age : INT **in** ]0,140] ; — - assertion a2
serviceMilitaire : STRING **in** $\{non, en-cours, fait, sursitaire, exempté\}$ ; - - assertion a3

[8] Bien que les notions de vue et de classe ne soient pas synonymes, nous suivons la proposition de E. Bertino dans [Bertino & Guerrini, 1995], qui propose de noter C la classe courante et C* la classe C et ses classes filles.

possede : **setOf** (VOITURE) ; - - VOITURE est une vue du P-type VEHICULE
**reverse** Proprietaire ;
**key** id - - Clé externe, base de la contrainte d'unicité
**methods ...** - - Spécification des autres fonctions
**assertions** - - Les assertions a1-a3 sont définies ci-dessus

(a4) $age < 18 \Longrightarrow$ serviceMilitaire = non
(a5) $age \geq 18$ **and** sexe = m $\Longrightarrow$ serviceMilitaire in $\{en-cours, fait, sursitaire, exempté\}$
(a6) sexe = f $\Longrightarrow$ serviceMilitaire = non ;

**end** ; - - Un attribut OID : toid est créé automatiquement dans la vue minimale

**view** ENSEIGNANT : PERSONNE - - ENSEIGNANT spécialise PERSONNE
**a**ttr revenu : FLOAT ;
affectation : UFR ;
grade : STRING ;
indice : INT ;
**assertions**
(a7) grade in $\{maitreConf, professeur, moniteur, ATER, secondDegr\}$ ;
(a8) indice $\leq 1200$ ;
**end** ENSEIGNANT ;

**view** ETUDIANT : PERSONNE
**attr** diplomes : setof STRING in $\{highschool, A.A., A.S., B.A., B.S., M.S., M.A., degree\}$
etudes : STRING in $\{graduate, postgraduate, doctorate\}$ ;
**end** ETUDIANT ;

**view** MONITEUR : ETUDIANT, ENSEIGNANT - - spécialise etudiant et enseignant
**assertions** - - et donc définit un sous-ensemble de leur
(a9) age $\leq 27$ - - intersection caractérisé par les assertions de la vue
(a10) statut = moniteur ;
(a11) etudes = these ; - - les attributs statut, etudes et diplomes
(a12) diplomes contain « maitrise2 » ; - - ont été définis dans la vue etudiant
. **end** ;

**view** SPORTIF : PERSONNE - - spécialise personne **attr** estSportif : Bool ;
**end** SPORTIF ;
On peut aussi définir les vues suivantes :
view ADULT : PERSONNE assertions $age \geq 18$ end ;
view MINEUR : PERSONNE assertions $age < 18$ end ;
view SENIOR : PERSONNE assertions $age \geq 65$ end ;
view FEMME : PERSONNE assertions $sexe = f$ end ;

qui permettent d'illustrer la fonction de classement : selon son âge ou son sexe, un individu sera automatiquement situé dans ces vues. Les vues ne sont pas exclusives les unes des autres, et un objet peut appartenir simultanément à plusieurs vues. La figure 4.2 ci-dessous présente d'une part la hiérarchie du P-type PERSONNE et d'autre part le diagramme d'inclusion ensembliste des ensembles d'intérêt des vues de ce P-type.

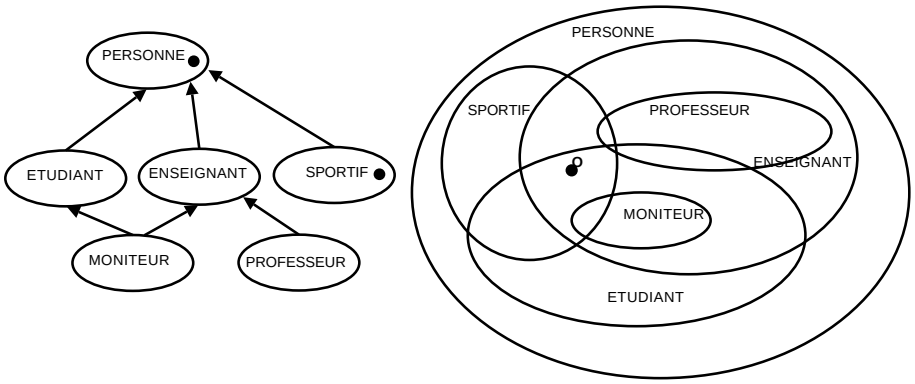


FIG. 4.2 – hiérarchie de spécialisation des vues et inclusions de leurs domaines

### 4.3.5 Espace de classement

Les Contraintes de Domaine Elémentaires (CDEs) et les Dépendances Inter-Attributs (DIAs) constituent ensemble des catégories de contraintes particulièrement étudiées en Osiris. Cette étude a donné naissance au concept d'*Espace de Classement* d'un P-type. Cet espace est un espace n-dimensionnel, où n est le nombre d'attributs classificateurs du P-type. La méthodologie utilisée pour générer l'Espace de Classement d'un P-type s'appuie sur la partition de l'espace de ses objets induite par l'ensemble des CDEs et des DIAs.

#### Définition

l'Espace de Classement d'un P-type est l'espace quotient de l'espace des objets relativement à la relation d'équivalence "*avoir la même valeur de vérité vis-à-vis de chacun des prédicats utilisés dans les DIAs de toutes les vues du P-type*".

#### Sous-Domaine Stable

Dans un P-type T, on considère pour chaque attribut $Attr_i$ l'ensemble P($Attr_i$ ) des prédicats élémentaires sur $Attr_i$ qui apparaissent dans les assertions de toutes les vues de T. Chaque prédicat élémentaire est de la forme $Attr_i \in D_{ik}$ où $D_{ik}$ est un sous-ensemble du domaine de définition $\Delta_i$ de $Attr_i$.

Un prédicat élémentaire $Attr_i \in D_{ik}$ détermine une *partition* de $\Delta_i$ en deux éléments : $D_{ik}$ et ($\Delta_i$-$D_{ik}$). Le produit de toutes les partitions [Stanat & McAllister, 1977] définies par les prédicats de P($Attr_i$) constitue une partition de $\Delta_i$ dont les *blocs* $d_{ij}$ sont appelés Sous-Domaines Stables (SDS).
Tout SDS vérifie la propriété de stabilité.
Propriété de stabilité d'un attribut : Lorsque, pour un objet $o_k$ , la valeur de l'attribut $Attr_i$ varie à l'intérieur d'un sous-domaine stable $d_{ij}$, l'objet continue de satisfaire exactement les mêmes prédicats de P($Attr_i$). Soit l'ensemble de DIAs définies dans toutes les vues du P-type PERSONNE (cf. Section 4.3.4).

(a1) sexe : char in {f, m} ;
(a2) age : int in ]0,140] ;
(a3) serviceMilitaire : string in $\{non, en-cours, effectué, sursitaire, exempté\}$;
(a4) age $<$ 18 $\Longrightarrow$ serviceMilitaire = non ;

(a5) age $\geq$ 18 and sexe = m $\Longrightarrow$ ServiceMilitaire in {$en - cours, effectué, sursitaire, exempté$};;
(a6) sexe = f $\Longrightarrow$ serviceMilitaire = non ;
(a7) $age \leq 27$ ;
(a8) statut= moniteur ;
(a9) etudes= these ;
(a10) diplômes contain DEA /*cas particulier d'attributs multivalués*/
(a11) age $\geq$ 18 ;
(a12) age $<$ 18 ;
(a13) age $\geq$ 65 ;
(a14) sexe = f ;

On notera que les trois premières assertions définissent le domaine des attributs *sexe*, *age* et *serviceMilitaire* dans le P-type PERSONNE, car elles sont définies dans sa vue minimale. Par contre, l'assertion (a9) définit uniquement le domaine de l'attribut *etudes* dans la vue MONITEUR. C'est pourquoi le domaine de définition d'*etudes* dans le P-type PERSONNE est « these » $\cup$ AUTRE $\cup$ UNDEFINI $\cup$ UNKNOWN, où AUTRE est un ensemble générique de chaines de caractères dont la valeur exacte est sans intérêt. Potentiellement, AUTRE contient toutes les chaînes de caractères différentes de « these ».

Les produits de partitions des attributs *age*, *serviceMilitaire*, *sexe* et *etude*, déterminées par leurs prédicats élémentaires, sont donnés ci-dessous :

age : $d_{11}$ = [0, 18[, $d_{12}$ = [18, 27], $d_{13}$ = ]27, 65[, $d_{14}$ = [65,140]
sexe : $d_{21}$ = {f}, $d_{22}$ = {m}
serviceMilitaire : $d_{31}$= {non}, $d_{32}$ ={$en - cours, effectué, sursitaire, exempté$}
etude : $d_{41}$= {these}, $d_{42}$ = autre, $d_{43}$ = indéfini

Un élément de la partition d'un attribut constitue un sous-domaine stable de l'attribut, noté SDS ou plus simplement d. Nous noterons par $SDS_{Attr}$ l'ensemble des sous-domaines stables de l'attribut Attr. Dans l'exemple en cours les SDS des différents attributs sont :

$SDS_{age}$ = { $d_{11}$, $d_{12}$, $d_{13}$, $d_{14}$ }
$SDS_{sexe}$ = { $d_{21}$, $d_{22}$ }
$SDS_{serviceMilitaire}$ = {$d_{31}$, $d_{32}$}
$SDS_{etude}$ = {$d_{41}$, $d_{42}$, $d_{43}$ }

**Eq-classe**

La partition du domaine de chaque attribut d'un P-type se prolonge en la partition de l'espace de ses objets, et constitue l'Espace de Classement du P-type. Ainsi, en considérant que les domaines des **n** attributs classificateurs d'un P-type ont été partitionnés en sous-domaines stables, l'Espace de Classement d'un P-type est un sous-ensemble du produit cartésien de ses ensembles SDS : $SDS_1$ * $SDS_2$ *...*$SDS_i$ *... *$SDS_n$ = {<$d_{1i}$, $d_{2j}$ , ...$d_{nk}$> |$d_{1i} \in SDS_1$, ...,$d_{nk} \in SDS_n$}
où, $SDS_j$ représente l'ensemble des Sous-Domaines Stables des $Attr_j$ , pour j $\in$ [1..n].
L'espace de classement est donc un espace à **n** dimensions, où chaque élément, noté *Eq-classe* (pour classe d'Equivalence), est un hypercube représenté par un n-uplet de n attributs. Lorsque des représentations graphiques sont nécessaires nous nous restreindrons à une représentation

basée sur trois attributs, car l'espace 3D est le plus grand espace visuellement compréhensible par un être humain. Ainsi, en considérant uniquement les trois attributs *sexe*, *age* et *serviceMilitaire*, l'Espace de Classement du P-type PERSONNE est représenté par :

Les Eq-classes valides du P-type sont celles qui satisfont les assertions de la vue minimale. Elles sont représentées en gras sur le dessin (cf. 4.3). Ainsi, l'Eq-classe ($d_{14}$, $d_{22}$, $d_{32}$), qui contient entre autres l'objet (age=70, sexe=m et serviceMilitaire = effectué) est valide, tandis que tout objet de l'Eq-classe ($d_{11}$, $d_{22}$, $d_{32}$) est invalide, car toute personne de moins de 18 ans (age $\in d_{11}$ ) ne peut que satisfaire $d_{31}$.

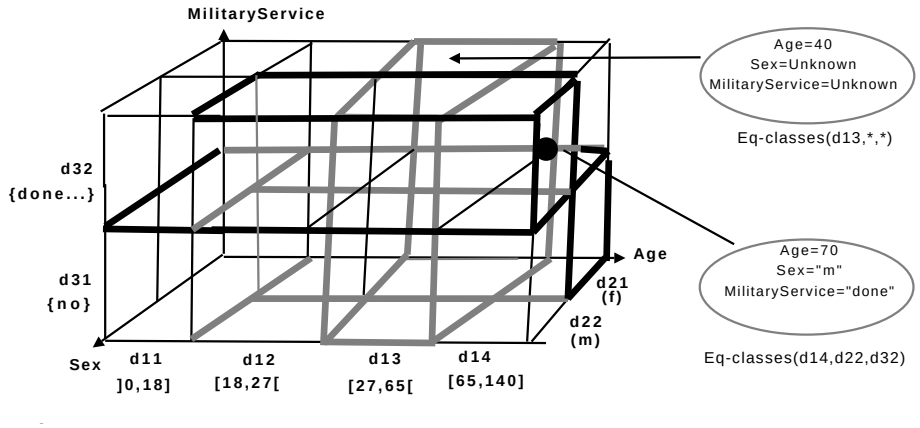


FIG. 4.3 – L'espace de classement du P-type *PERSONNE*

La propriété de stabilité d'un attribut se généralise à l'Espace de Classement :
Propriété de stabilité d'une Eq-classe : tous les objets d'une même Eq-classe ont la même validité vis-à-vis de toutes les vues du P-type.

Corollaire 1 : lorsqu'un seul attribut d'un objet est modifié tout en restant dans le même Sous-Domaine Stable, l'objet continue de satisfaire les mêmes prédicats, donc les mêmes assertions, et en conséquence les mêmes vues.

Corollaire 2 : lorsque plusieurs attributs d'un objet sont modifiés tout en restant chacun dans le Sous-Domaine Stable initial, l'objet continue de satisfaire les mêmes prédicats, donc les mêmes assertions, et en conséquence les mêmes vues.
Comme deux objets de la même eq-classe satisfont les mêmes assertions, et donc valident (ou invalident) les mêmes vues, il est possible de déterminer a priori les vues que les objets d'une Eq-classe satisfont. En conséquence, on peut associer à chaque vue les Eq-classes qui la valident.

Nombre d'Eq-classes
Soit **n** le nombre d'attributs classificateurs d'un P-type **T**. En considérant que chaque attribut a **p** Sous-Domaines Stables, le nombre maximal d'Eq-classes de **T** est $p^n$
Etant donné l'explosion combinatoire des Eq-classes d'un P-type, la relation Eq-classe/vue n'est pas maintenue statiquement en Osiris. En pratique, seules les Eq-classes qui contiennent au moins un objet sont rendues persistantes.

### Semi-domaine ou Eq-classe généralisée

La notion de partage sous-tend la manipulation d'objets ayant des valeurs manquantes : l'utilisateur qui crée un objet connaît au plus les données spécifiques aux vues au travers lesquelles il le perçoit. C'est pourquoi il est normal que certains objets présentent, à des moments particuliers de leur vie, des valeurs inconnues. Dans de tels cas, pour tout attribut dont la valeur est inconnue, le sous-domaine stable est lui aussi inconnu. Afin de manipuler de tels objets, le système Osiris a défini un sous-domaine stable dont le domaine se confond avec le domaine de l'attribut lui-même.

Un *semi-domaine* ou *Eq-classe* généralisée du P-type T caractérisé par n attributs classificateurs est un n-uplet $<d_{1f}, d_{21}, d_{ik}, \ldots, d_{np}>$ tel que $\exists$ i| $d_{iq}=\Delta_i$ lui-même, noté "*", pour i $\in$ [1, n], où $\Delta_i$ représente le domaine du $i_{ime}$ attribut de T.
Une composante explicite d'un semi-domaine est un élément différent de "*". Un semi-domaine dont tous les éléments sont des composantes explicites est une Eq-classe.

### Détermination des Eq-classes

Nous avons montré précédemment que tous les objets d'une Eq-classe donnée avaient la même valeur de vérité pour toutes les assertions d'un P-type. De fait, elles ont la même valeur de vérité pour toutes les assertions spécifiques à une vue. Nous constatons donc qu'une vue est soit valide, soit invalide pour une Eq-classe donnée. Dire qu'une vue est valide pour une Eq-classe signifie que tous les objets qui lui sont associés appartiennent à cette vue. Inversement, si la vue est invalide pour l'Eq-classe, les objets qui lui sont associés n'appartiennent pas à cette vue.

Nous allons illustrer ceci à partir du P-type PERSONNE défini ci-dessus : Les domaines des attributs *sexe* et *age* sont patitionnés de la façon suivante :
Sous-domaines stables de l'attribut *age* :

$d_{11}$=]0,18[,$d_{12}$=[18,27],$d_{13}$=]27,65[,$d_{14}$=[65,140]

Sous-domaines stables de l'attribut *sexe* :

$d_{21}$=f, $d_{22}$= m

La figure 4.4 établit pour les vues (FEMME, HOMME, MINEUR, ADULT et SENIOR) définies dans le schéma de le P-type PERSONNE, la cartographie de sa validité par rapport à chaque Eq-classe.

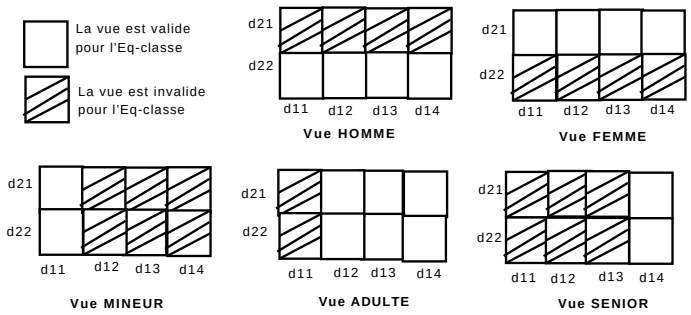


FIG. 4.4 – Cartographie de la validité des vues

### 4.3.6 Classement d'objets

Le classement d'un objet dans les vues de son P-type se fait au travers d'un réseau appelé réseau de classement.

#### Réseau de principe

Le réseau de classement d'un P-type est un réseau connexionniste à trois couches et sans apprentissage. Les cellules de la première couche représentent les sous-domaines stables du P-type, les cellules de la deuxième couche représentent les Eq-classes, et les cellules de la troisième couche représentent les vues du P-type. D'un point de vue statique, l'architecture du réseau de classement est le suivant :

1. chaque cellule sous-domaine stable est connectée à toutes les cellules Eq-classes dont ce sous-domaine stable est un composant.
2. chaque cellule Eq-classe :
   - est connectée, en entrée, à un ensemble de sous-domaines stables, un par attribut classificateur du P-type,
   - implémente un ET logique de ses entrées,
   - est connectée, en sortie, à toutes les cellules des vues qu'elle valide.
3. chaque cellule Vue :
   - est connectée, en entrée, à toutes les Eq-classes qui la valident,
   - réalise un OU logique de ses entrées,
   - rend un résultat booléen.

Lors d'un classement,

1. les cellules représentant un sous-domaine stable valide ont une valeur de sortie égale à *vrai* ; les autres ont une valeur de sortie égale à *faux*,
2. la cellule Eq-classe dont toutes les entrées sont à vrai a une valeur de sortie à *vrai* ; toutes les autres ont une valeur de sortie à *faux*,
3. les cellules Vue dont au moins une entrée est a une valeur *vrai*, ont une valeur de sortie à *vrai*. Les autres ont une valeur de sortie à *faux*.

Soit un P-type **T** ayant **n** attributs classificateurs et **v** vues. Si chaque attribut a **p** sous-domaines stables,

1. le nombre de cellules sous-domaines stables est égal à (n x p),
2. le nombre d'Eq-classes de T est potentiellement égal à $p^n$,
3. le nombre de cellules « vue » est égal à v.

Ce réseau de principe a donc une taille de (n x p) + $p^n$ + v. Il n'est donc pas utilisable en pratique, à la fois pour des raisons de taille (le nombre de cellules est exponentiel en fonction du nombre d'attributs), et de temps d'exécution, du moins dans le paradigme d'une programmation séquentielle.

#### Méthodologie de classement

En Osiris, classer un objet dans un P-type revient à classer son Eq-classe. La méthodologie utilisée est la suivante :

1. pour chaque attribut valué, on détermine le sous-domaine stable correspondant,
2. on déduit l'Eq-classe de l'objet [9] ,

[9] Ou les Eq-classes de l'objet, si celui-ci n'est pas complètement valué

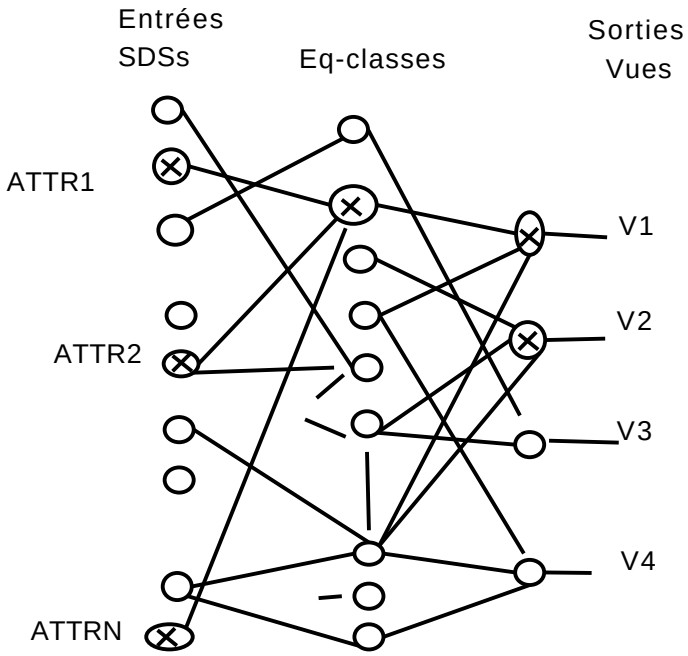


FIG. 4.5 – réseau de principe pour le classement

3. on classe le n-uplet ainsi obtenu en utilisant le réseau de classement.

### Classement d'objets complets : classement VI

Un objet complet, ou objet complètement valué, est un objet pour lequel on connaît la valeur de tous les attributs classificateurs de son P-type. Dans un tel cas, pour une vue $V_i$ le résultat du classement est soit Valide soit Invalide.

Soit **O** un objet dont les valeurs des attributs $ATTR_1$ , $ATTR_2$, ..., $ATTR_n$ se projettent (appartiennent) respectivement sur les sous-domaines stables $d_{12}$ , $d_{22}$, ..., $d_{n2}$ (cf. 4.5). Son Eq_classe est donc $< d_{12}$ , $d_{22}$, ..., $d_{n2}>$. En utilisant le réseau de la figure 4.5, on peut conclure que cet objet est valide pour les vues V1 et V2, et invalide pour les vues $V_3$ et $V_4$. Ces conclusions sont identiques pour tous les objets appartenant à cette même Eq-classe.

Pour notre exemple ci-dessus, la table 5.1 montre le classement d'objets complets avec les deux attributs classificateurs *age* et *sexe*.

Par exemple, considérons les objets :

```
O1= (nom: Dupond, sexe: M, dateNaissance: (jour:10, mois: 01, annee:1956), age: 42).
O2= (nom: Durant, sexe: M, dateNaissance: (jour:01, mois: 08, annee:1978), age: 29).
```

Ils ont pour Eq-classe ($d_{13}$ ,$d_{22}$) et appartiennent donc uniquement aux vues HOMME et ADULTE.

### Classement d'objets incomplets : classement VIP

Un objet **O** est dit incomplet lorsque la valeur d'au moins un attribut classificateur est inconnue. Dans le mode *base de données*, lorsque l'utilisateur affecte l'objet à l'une de ses vues,

| **Eq-classe** | **FEMME** | **HOMME** | **MINEUR** | **ADULTE** | **SENIOR** |
|---|---|---|---|---|---|
| $[d_{11}$ ,$d_{21}]$ | Invalide | Valide | Valide | Invalide | Invlide |
| $[d_{11}$ ,$d_{22}]$ | Valide | Invalide | Valide | Invalide | Invlide |
| $[d_{12}$ ,$d_{21}]$ | Valide | Invalide | Invalide | Valide | Invlide |
| $[d_{12}$ ,$d_{22}]$ | Invalide | Valide | Invalide | Valide | Invalide |
| $[d_{13}$ ,$d_{21}]$ | Valide | Invalide | Invalide | Valide | Invalide |
| $[d_{13}$ ,$d_{22}]$ | Invalide | Valide | Invalide | Valide | Invalide |
| $[d_{14}$ ,$d_{21}]$ | Valide | Invalide | Invalide | Valide | Invalide |
| $[d_{14}$ ,$d_{22}]$ | Invalide | Valide | Invalide | Valide | Valide |

TAB. 4.1 – Les vues et les Eq-classes

en général, les attributs sans intérêt pour ces vues ne sont pas - et ne peuvent pas - être valués, car a priori l'utilisateur n'a même pas à connaitre leur existence. De plus, certains attributs des vues auxquelles l'utilisateur a droit peuvent avoir une valeur inconnue, pourvu que le qualifieur *unknown* leur ait été attribué.

Lorsqu'un objet **O** n'est pas complètement valué, l'objet n'appartient pas à une seule Eq-classe, mais potentiellement à un certain nombre d'Eq-classes. L'ensemble de ces Eq-classes définissent son semi-domaine. Pour prendre en compte les objets incomplets, une adaptation de la méthodologie de classement et du réseau de classement a été faite [BASSOLET, 1992]. Le résultat est la classification VIP où P signifie Potentiel. Les vues potentielles sont les vues qui ne sont ni valides (aucun des attributs connus ne contredit les contraintes de la vue), ni invalide (les attributs connus ne sont pas suffisants pour montrer que l'ensemble de contraintes de la vue est logiquement valide). Une vue potentielle peut devenir valide ou invalide pendant le cycle de vie de l'objet quand quelques valeurs inconnues deviennent connues. La détermination des résultats du classement VIP est faite par l'utilisation d'une approche probabiliste. En conséquence, une vue $V_i$ d'un P-type est :

1. **V**alide. Si la valeur de probabilité calculée pour $V_i$ est égale à 1 : l'objet appartient de façon sûre à la vue,
2. **I**nvalide. Si la valeur de probabilité calculée pour $V_i$ est égale à 0 : de façon certaine, l'objet n'appartient pas à $V_i$,
3. **P**otentielle. Dans tous les autres cas l'objet appartient potentiellement à la vue $V_i$ .

En Osiris, le résultat final du classement d'un objet pour une vue est un élément de l'ensemble $\{V, I, P\}$ et le classement est dénommé classement VIP. De même que dans le cas des Eq-classes, tous les objets appartenant à un semi-domaine satisfont (/ ne satisfont pas /satisfont potentiellement) exactement les mêmes vues.

Dans la version actuelle d'Osiris, le classement d'un objet donne pour chaque vue une valeur du domaine $\{V, I, P\}$, où V, I, P symbolisent respectivement les valeurs **V**alide, **I**nvalide et **P**otentiel.

Par exemple, l'objet O3= (nom : Martin, sexe : ?, date_naissance :(jour : 25, mois : 05, annee : 2005, age :4) peut potentiellement appartenir à l'Eq-classe ($d_{11}$ ,$d_{21}$) et à l'Eq-classe ($d_{11}$ ,$d_{22}$). On peut malgré tout déduire de la cartographie précédente que, de manière certaine, il appartient à la vue MINEUR et n'appartient pas aux vues ADULT et SENIOR. Il appartient potentiellement à la vue FEMME et à la vue HOMME. Les carrés représentés en blanc dans la figure 4.6 constituent l'ensemble des Eq-classes potentielles de cet objet.

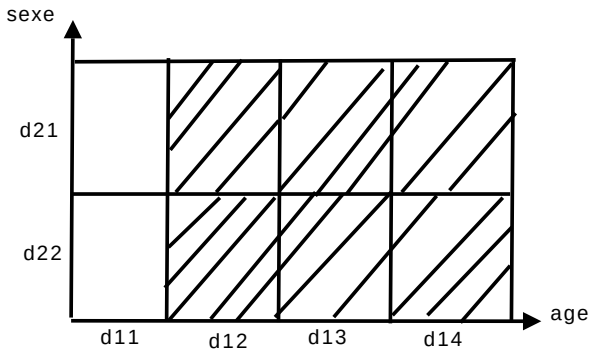


FIG. 4.6 – Eq-classes potentielles pour l'objet O3 dont l'attribut classificateur *sexe* n'est pas connu

Le figure 4.7 montre les vues potentielles et valides pour cet objet.

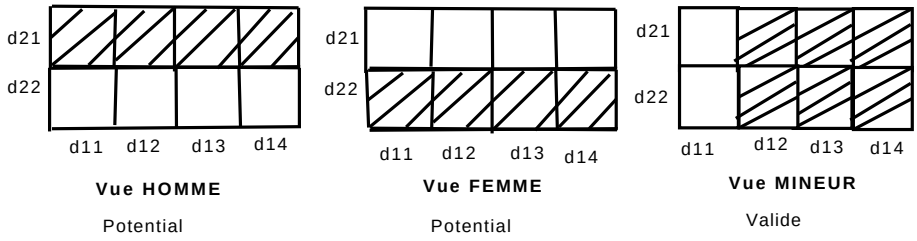


FIG. 4.7 – Vues potentielles et valides pour l'objet O3 dont l'attribut classificateur *sexe* n'est pas connu

### Avantages du classement d'Osiris

Par rapport aux algorithmes classiques de classement dans les bases de connaissances, le classement en Osiris présente un certain nombre d'avantages, parmi lesquels nous soulignons :

1. l'explicitation des vues invalides d'un objet. Cette information est très importante pour l'aide à décision, surtout lorsque les systèmes en jeu sont dans des domaines critiques, tels que la médecine, la biologie ou le nucléaire. Par exemple, en cas d'intoxication par des champignons, savoir que la vue/classe VENENEUX est exclue peut être plus important que de connaître la vue/classe exacte de l'objet. De même, savoir qu' à un instant donné, une centrale nucléaire ne court pas de risque majeur est plus important que de savoir la vue/classe exacte correspondant à sa situation courante,
2. la déduction des vues potentielles. Cette déduction se fait sans que l'utilisateur soit amené à intervenir explicitement afin d' évaluer certains attributs. En conjonction avec la fonction de comparaison de vues d'Osiris, ce résultat permet de circonscrire les examens et tests complémentaires à réaliser pour déterminer la vue/classe exacte d'un objet,
3. la possibilité de déduire la validité d'une vue, même en présence d'un objet incomplet. Ce cas de figure se présente lorsque toutes les Eq-classes d'un semi-domaine sont valides pour une vue donnée. Par exemple, la vue ENFANT est valide pour toute personne de moins de dix-huit ans, même si on ne connaît pas son sexe.

### 4.3.7 Indexation d'objets

Une structure d'indexation nommée ISD (Indexing Structure Descriptor) est définie pour chaque P-type [Simonet et al., 94]. Ses principaux champs sont :

1. un vecteur de sous-domaines stables représentant une Eq-classe ou un semi-domaine,
2. un vecteur de vues donnant le statut (Valide, Invalide, Potentiel) de chaque vue du P-type pour l'ensemble des objets indexés par cet ISD,
3. une référence à l'ensemble des objets de l'ISD,
4. le nombre total d'objets indexés par cet ISD.

Lorsqu'un objet est créé, il est classé. Le classement nécessite la prédétermination des sous-domaines stables des attributs classificateurs du P-type de l'objet. Lorsque la valeur d'un attribut classificateur $ATTR_i$ est inconnue, Osiris lui affecte son sous-domaine stable généralisé qui couvre le domaine complet d'$ATTR_i$. Une fois les sous-domaines stables déterminés, l'objet est classé et Osiris détermine pour chacune des vues du P-type sa validité. Ainsi, une vue sera dite :

- **V**alide si toutes ses assertions sont vérifiées,
- **I**nvalide si au moins une de ses assertions est contredite,
- **P**otentielle, s'il manque des données pour déduire sa validité ou invalidité.

A noter que des vues potentielles ne peuvent exister que si l'on gère des objets incomplets. Une fois l'objet classé, Osiris dispose de toute l'information pour déterminer son ISD et l'objet peut y être rangé. Si l'objet constitue la première instance dans son ISD, celui-ci est lui-même créé. Lorsqu'un objet est modifié, il reste dans le même ISD si aucun attribut ne change de sous-domaine stable. Autrement, il change d'ISD.

On remarquera que lorsque des objets incomplets sont à gérer, deux ISD distincts peuvent partager certaines Eq-classes. Ainsi, si l'on représente par $d_{i*}$ le sous-domaine généralisé de l'attribut $ATTR_i$, les ISDs :

$ISD_{12}$ : <$d_{11}$, $d_{23}$ , $d_{35}$, V, V, I, V>

et

$ISD_{20}$ : <$d_{11}$, $d_{2*}$ , $d_{35}$, V, P, I, P>

se partagent l'Eq-classe <$d_{11}$, $d_{23}$ , $d_{35}$>. Lorsqu'un attribut inconnu devient connu, l'objet change d'ISD et son nouvel ISD est obligatoirement subsumé par l'ISD initial.

Exemple : Considérons les trois objets O1, O2, O3 cités en exemple ci-dessus.

Pour les deux objets :

O1= (nom : Dupond, sexe : M, age : 42)

O2= (nom : Durant, sexe : M, age : 29)

Ils ont pour Eq-classe ISD01 :([$d_{13}$ ,$d_{22}$]).

On montre dans la table 5.2 l'indexation des objets O1 et O2 :

| **Eq-classe** | **FEMME** | **HOMME** | **MINEUR** | **ADULTE** | **SENIOR** |
|---|---|---|---|---|---|
| [$d_{13}$ ,$d_{22}$] | Invalide | Valide | Invalide | Valide | Invalide |
| ISD01 | I | V | I | V | I |

TAB. 4.2 – Indexation des objets O1 et O2

pour l'Objet O3=(nom : Martin, sexe : ?, age : 4)

Il a pour Eq-classe : ISD02 ([$d_{11}$ ,$d_{21}$] ,[$d_{11}$ ,$d_{22}$]).

On montre dans la table 4.3 l'indexation de l'objet O3 :

| Eq-classe | FEMME | HOMME | MINEUR | ADULTE | SENIOR |
|---|---|---|---|---|---|
| $[d_{11}, d_{21}]$ | Invalide | Valide | Valide | Invalide | Invlide |
| $[d_{11}, d_{22}]$ | Valide | Invalide | Valide | Invalide | Invalide |
| ISD02 | P | P | V | I | I |

TAB. 4.3 – Indexation de l'objet O3

La figure 4.8 représente la structure d'indexation ISD pour les objets O1, O2 et O3. Elle répresente les Eq-classes créées lors de l'insertion d'objets (O1, O2, 03). Chaque OID pointe sur une entrée du dictionnaire de données, qui est la structure qui contient les objets.

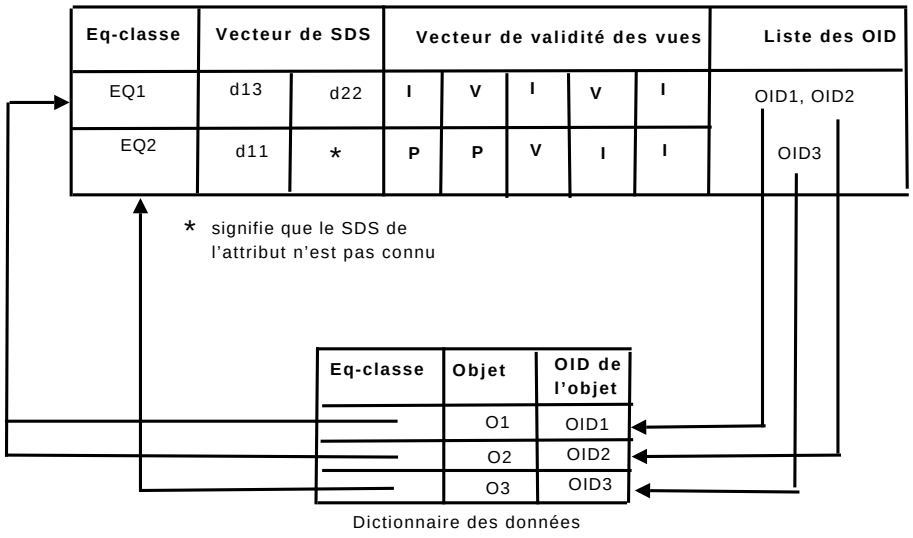


FIG. 4.8 – La structure d'indexation pour les objets O1,O2,O3

### 4.3.8 Optimisation de requêtes en Osiris

L'optimisation de requêtes consiste à transformer une requête en une requête équivalente basée sur un espace de recherche plus restreint. Une requête est dite équivalente à une autre requête si et seulement si son évaluation aboutit au même résultat que la requête initiale, et cela pour tout état de la base de données.

#### Forme générale des requêtes

Osiris distingue les requêtes d'interrogation des requêtes de création de nouveaux P-types à partir de P-types existants. Contrairement aux requêtes de création de nouveaux P-types qui sont dites *object-generating*, les requêtes d'interrogation sont *object-preserving* [Scholl et al., 1991]. Dans ce mémoire nous nous intéressons uniquement aux requêtes d'interrogation. La forme générale d'une requête est :

(Contexte | Condition) $[Attr_i]^{0,*}$ où,

– *Contexte* est une Contrainte sur les Vues d'un P-type,

- *Condition* est une condition logique sur les attributs des vues de la partie *contexte* [10]. La forme de la condition est identique à celle des CDEs et des DIAs,
- le résultat d'une requête est soit une collection d'Oids, soit une collection de n-uplets <Oid, $Attr_i$, $Attr_j$, ...>, où $Attr_i$, $Attr_j$, ..., sont les attributs nommés de $[Attr_i]^{0,*}$ .

Des exemples de requêtes sont :

(ETUDIANT **and** SPORTIF | Age > 23)
(ENSEIGNANT **or** SPORTIF | Age > 40 **and** Revenu > 10 000) [nom, age]
(MONITEUR **and not** SPORTIF)
(ETUDIANT **or not** SPORTIF)

Lorsqu'une requête est étiquetée, elle définit une **vue dynamique**. Contrairement aux vues définissant un P-type, les contraintes définies dans la partie *condition* d'une vue dynamique ne permettent pas de partitionner l'espace des objets d'un P-type (qui est défini statiquement).

### Optimisation sémantique des requêtes

L'optimisation est dite sémantique si la transformation de la requête s'appuie sur des connaissances sémantiques associées aux classes du schéma conceptuel de la base de données. Dans une base de données, ces connaissances sont essentiellement exprimées par des contraintes (d'intégrité) du schéma conceptuel.

### Optimisation sémantique : méthodologie Osiris

En Osiris, une requête est évaluée à l'intérieur d'un P-type donné. Elle est assimilée à une vue dynamique du P-type et n'est pas classée explicitement par rapport aux autres vues du P-type, qu'elles soient statiques ou dynamiques. Par contre, elle est réécrite en termes des sous-domaines stables des attributs qu'elle contient, ce qui revient à la situer dans l'espace de classement en termes d'Eq-classes (génériques) valides, invalides et potentielles.

Par exemple, soit le P-type PERSONNE (voir exemple) et la requête (PERSONNE | age < 15). En classant la requête dans la hiérarchie de vues, l'optimisation sémantique classique de la requête permettrait de restreindre la recherche des objets d'intérêt aux objets de la vue MINEUR.

Cependant, en Osiris, la recherche du sous-espace englobant minimal d'une requête se fait de manière plus ciblée en utilisant les ISD et non pas les vues. Lors de l'évaluation d'une requête, notre enjeu est donc de déterminer :

1. les ISDs dont les objets font, de manière certaine, partie de la réponse.
2. les ISDs dont il faut examiner les objets individuellement.
3. les ISDs dont les objets ne font pas partie de la réponse.

La méthodologie mise en place en Osiris pour l'optimisation sémantique des requêtes s'appuie :

1. d'abord sur la détermination des ISDs *propres* à la requête. A chacun des ISDs propres est affectée une valeur de vérité **VS** ou **VP** pour **V**aleur **S**ûre et **V**aleur **P**otentielle. Un ISD est à valeur VS si et seulement si tous ses objets satisfont la requête ; il sera à valeur VP si certains de ses objets peuvent ne pas satisfaire l'ISD.

[10] Une algèbre des vues a été définie, précisant quels sont les attributs et les assertions d'une vue construite par généralisation(V1 **ou**, par spécialisation V2 (V1 **and** V2, et par complémentation (**not** V2)

2. puis sur l'extraction des objets des ISDs *effectifs* [11], ISDs effectifs qui subsument les ISDs propres de la requête.

Par exemple, soient les SDSs de l'attribut age : age : $d_{11}$ = [0, 18[, $d_{12}$ = [18, 27], $d_{13}$ = ]27, 65[, $d_{14}$ = [65,140] et la requête : (ETUDIANT | age < 23)

Les ISDs propres de cette requête sont {<($d_{11}$, *, *)(1, 1, *, *, ...*), VS>, <($d_{12}$, *, *)(1, 1, *, *, ...*), VP>}. Lors de l'évaluation d'une telle requête, et en fonction des ISDs présents dans le SI, les ISDs effectifs sont ceux se projetant sur l'un ou l'autre des ISDs propres. Selon que l'ISD propre est un ISD à valeur de vérité VS ou VP, les objets d'un tel ISD effectif satisfont la réponse ou doivent être filtrés par la condition de la requête. Ainsi, en considérant que l'ISD <($d_{11}$, $d_{12}$, *) (1, 1, 1, 0, ...)...> existe, cet ISD est bien un ISD effectif de la requête car il se projette bien sur l'ISD propre <($d_{11}$, *, *)(1, 1, *, *, ...*), VS>. De plus, comme la valeur de vérité de l'ISD propre en question est égale à VS, tous les objets de l'ISD effectif appartiennent à la réponse.

**ISDs propres d'une requête**

La détermination des ISDs *propres* à une requête se fait en quatre étapes :

Etape1 : transformation d'une requête en une disjonction de sous-requêtes conjonctives. Une requête est dite conjonctive si et seulement si, dans sa partie condition, le seul opérateur utilisé est l'opérateur **and**, et les opérandes sont soit des prédicats élémentaires sur les attributs ou sur les vues, soit la négation de prédicats élémentaires.
Pour chaque sous-requête sous forme conjonctive nous appliquons les étapes 2 à 4.

Etape2 : détermination des Eq-classes des ISDs d'intérêt.
Dans cette étape on détermine les Eq-classes appartenant aux ensembles **V**alide, **I**nvalide et **P**otentiel. Ces Eq-classes sont souvent des semi-domaines obtenus de la manière suivante :

1. pour tout attribut explicitement nommé dans la requête, déterminer l'ensemble des sous-domaines stables qui sont valides, l'ensemble de ceux qui sont potentiels, et l'ensemble de ceux qui sont invalides,
2. construire les Eq-classes, ou plus généralement les semi-domaines, d'intérêt pour la requête. Cette construction est le résultat du produit cartésien des sous-domaines stables valides et potentiels obtenus en 1),
3. pour chaque Eq-classe obtenue dans l'étape 2, lui affecter une valeur de vérité. Cette valeur de vérité est obtenue par la conjonction des valeurs de vérité des sous-domaines stables qui composent l'Eq-classe.

Ainsi,
- une Eq-classe est valide si et seulement si tous les sous-domaines stables qui la composent sont valides pour la requête. Cette Eq-classe appartient à l'ensemble ***VS***,
- une Eq-classe est potentielle si et seulement il existe au moins un sous-domaine stable potentiel parmi ses sous-domaines stables. Cette Eq-classe appartient à l'ensemble ***VP***,

Etape 3 : détermination du vecteur de vues des ISDs d'intérêt.
Le vecteur de vues d'une conjonction de vues est tout simplement un vecteur, noté VECT-VUE, où l'on a attribué une valeur de vérité (V ou F) aux vues explicitement nommées, la valeur de

[11] Un ISD effectif du SI est un ISD qui contient au moins un objet du SI à un moment donné

vérité des autres vues étant sans intérêt. La valeur de vérité attribuée à une vue explicitement nommée dans la requête est :

1. Valide si la vue est nommée dans sa forme affirmative,
2. Invalide, si la vue est nommée dans sa forme négative.

Etape 4 : construction des ISDs propres :
Ces ISDs sont obtenus par le produit cartésien des éléments de (VS ∪ VP) (étape2) et du vecteur de vues (étape3). On les dénote par ISDs propres. A tout ISD propre on attribue la valeur de vérité :

1. Valide, si son Eq-classe appartient à l'ensemble VS,
2. Potentiel, si son Eq-classe appartient à l'ensemble VP.

### Extraction des objets

La recherche des objets satisfaisant une requête se fait en trois étapes :

1. recherche des ISDs propres,
2. recherche des ISDs **effectifs** qui sont subsumés par les ISDs de la requête, où ISD est dit **effectif** s'il contient au moins un objet,
3. pour chaque ISD effectif, extraction automatique (s'il est est subsumé par un ISD valide) ou semi-automatique (s'il est subsumé par un ISD potentiel) des objets de la vue.

### Exemples de requêtes d'interrogation

Les exemples ci-dessous ne présentent que l'aspect « requête d'interrogation »

Exemple1
soit la requête :
Q : (PERSONNE | 12<Age<40)
et les sous-domaines stables de l'attribut Age :
Age : $d_{11}$ = [0, 18[, $d_{12}$ = [18, 27], $d_{13}$ = ]27, 65[, $d_{14}$ = [65,140]
Soit ***E*** l'ensemble des Eq-classes valides du P-type PERSONNE. La requête ci-dessus divise ***E*** en trois collections d'Eq-classes :
**VS** = ($d_{12}$, *,*,*,*) : les personnes dont l'âge est entre 18 et 27 satisfont la requête.
**VP** = ($d_{11}$, *,*,*,*), ($d_{13}$, *,*,*,*) : les personnes dont l'âge est soit inférieur à 18 soit supérieur à 27 sont à tester.
A noter que l'Eq-classe généralisée ($d_{14}$, *,*,*,*), constituée de personnes d'âge supérieur à 65, ne satisfait pas la requête.

Le vecteur de vues de Q est $VECTVUE_Q = (V, *, *, *, *, *, *, *, *)$. On peut remarquer qu'il est sans intérêt dans la mesure où la vue d'intérêt de Q est la vue minimale, vue à laquelle appartiennent obligatoirement tous les objets de ce P-type.
Les ISDs propres de Q sont donc :

1. ISDs propres Validité Sure : ((V, *, *, *, *, *, *, *, *), ($d_{14}$, *,*,*,*)),
2. ISDs propres Validité Potentiels : ((V, *, *, *, *, *, *, *, *), ($d_{11}$, *,*,*,*)) ; ((V, *, *, *, *, *, *, *, *), ($d_{13}$, *,*,*,*)).

Les objets des ISDs effectifs subsumés par les ISDs propres Valides font partie de la réponse sans qu'il soit besoin de les tester individuellement, tandis que ceux appartenant aux ISDs

subsumés par les ISDs propres Potentiels doivent être testés.

Exemple 2 : soit la requête :
Q1 : (MINEUR | 12<Age<40)
Les ensembles des Eq-classes ***VS, VP*** sont identiques à ceux précédemment sélectionnés. Par contre, $VECTVUE_{Q1}$ est : (*, *, *, *, *, *, V, *, *), où on a considéré que MINEUR est la 7ème vue [12] des neuf vues du P-type personne (cf.Section 4.3.4). En conséquence :

1. ISDs propres Valides : ((*, *, *, *, *, *, V, *, *), ($d_{14}$, *,*,*,*)),
2. ISDs propres Potentiels : ((*, *, *, *, *, *, V, *, *), ($d_{11}$, *,*,*,*)) ; ((V, *, *, *, *, *, *, *, *), ($d_{13}$, *,*,*,*)).

On remarque qu'aucun ISD effectif ne peut être subsumé par l'ISD potentiel (($d_{13}$, *,*,*,*), (*, *, *, *, *, *, V, *, *)), car MINEUR est incompatible avec $d_{13}$. Osiris ne recherche pas a priori à mettre en valeur cette incompatibilité, car cet ISD propre et tous les ISDs effectifs qu'il subsume sont exclus d'office dans la mesure où ils ne peuvent être l'ISD d'aucun objet valide du P-type.

## 4.4 Conclusion

Dans ce chapitre nous avons présenté le système Osiris, système qui implémente le modèle des P-types, et ses concepts de base. Nous nous sommes intéressés à l'utilisation du concept de P-type dans l'intégration de sources hétérogènes de données. Dans notre travail, nous nous intéressons d'une part au concept de vue qui, est central pour le modèle des P-types et qui se révèle être un concept clé pour l'intégration de documents XML, et d'autre part au système d'indexation offert par Osiris, qui permet l'optimisation sémantique de requêtes.

Dans le chapitre suivant, nous présentons notre approche, nommée OSIX (Osiris-based System for the Integration of XML documents), qui est basée sur le système Osiris et conçue pour l'intégration des documents XML. Nous montrons comment on peut étendre le système Osiris pour prendre en compte les documents XML et nous montrons la possibilité d'adapter le modèle Osiris à l'intégration de documents XML.

Notre approche utilise l'indexation proposée par OSIRIS afin d'optimiser le traitement des requêtes sur une source qui contient un ensemble de documents XML.

[12] l'ordre des vues est sans intérêt en soi.

# Chapitre 5

# Intégration des données XML en utilisant l'indexation proposée par Osiris : le Système OSIX

## 5.1 Introduction

Dans ce chapitre nous proposons l'approche OSIX (*Osiris-based System for the Integration of XML documents*), conçue pour l'intégration des données semi-structurées. Cette méthode s'appuie sur le système Osiris, système de représentation de données et de connaissances de la famille des Logiques de Descriptions (DL)(cf. chapitre 4). En particulier, nous nous intéressons d'une part au modèle de données d'Osiris, qui permet de définir plusieurs points de vues sur une même famille d'objets et d'autre part à son système d'indexation, qui permet l'optimisation sémantique des requêtes. En effet, une source XML peut contenir un grand nombre de documents XML avec des schémas plus au moins différents. Avec le modèle de données d'Osiris, il n'est pas nécessaire qu'un schéma XML corresponde à toutes les vues d'un P-type, mais seulement à un sous-ensemble de ses vues. Une conséquence immédiate est que des schémas XML très différents peuvent être en correspondance avec le même P-type. Par ailleurs, Osiris offre un système d'indexation multicritères puissant.

L'objectif de notre travail d'intégration de données XML est l'optimisation sémantique de requête en utilisant l'indexation d'un objet en Osiris.

Dans notre système, l'indexation d'une entité d'un document XML est basée sur les vues qu'il satisfait et par les SDSs auxquels appartiennent les valeurs atomiques contenues dans le document. En utilisant cette indexation on peut déterminer les documents XML qui satisfont une ou plusieurs vues. En conséquence, l'espace de recherche pour obtenir les documents satisfaisant une requête peut être réduit à l'espace de vues satisfaisant cette requête. Dans la section suivante on décrit l'architecture du système OSIX.

## 5.2 Architecture du système OSIX

L'architecture du système OSIX pour l'intégration de données semi-structurées(XML) est décrite dans la figure 5.1. Une étude a également été faite pour intégrer des sources des données structurées avec une approche hybride basée sur le système d'indexation d'Osiris [Kermanshahani, 2008].

Le système OSIX est composé de :

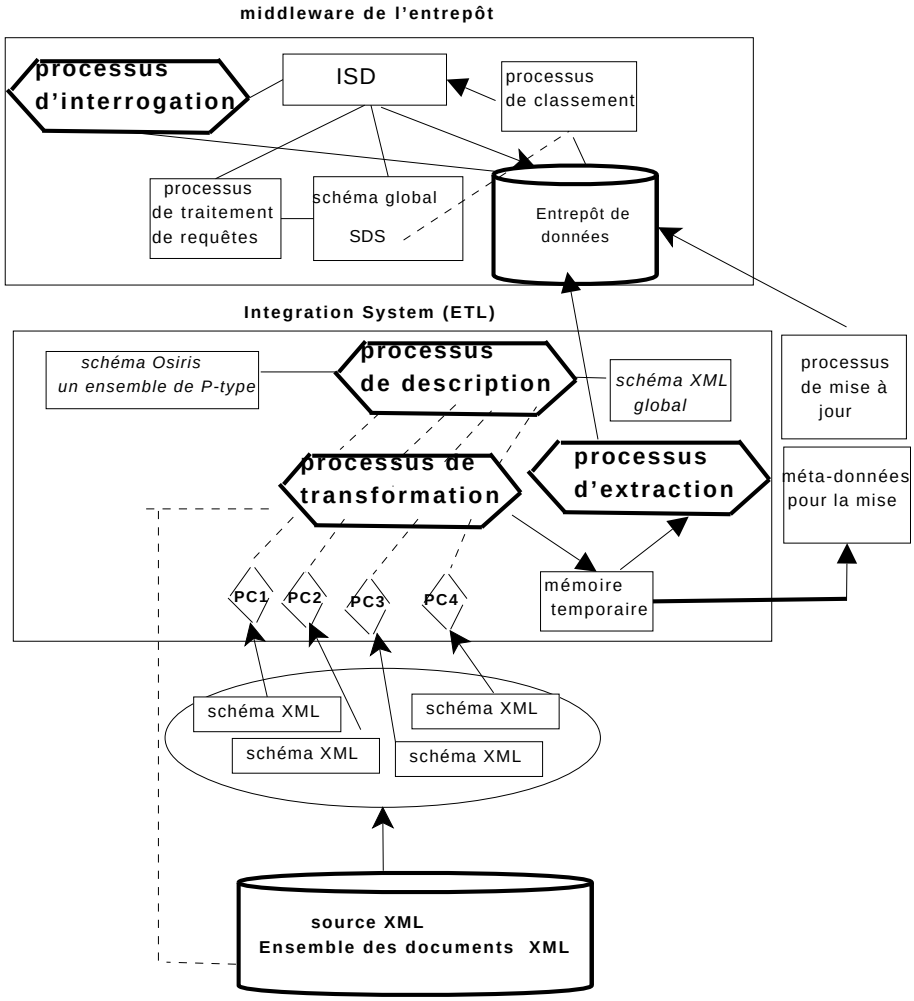


FIG. 5.1 – L'architecture de système OSIX

- un processus de description qui décrit un schéma Osiris sous la forme d'un schéma XML ;
- un processus de correspondance (PC) pour chaque schéma concret XML, qui effectue la correspondance entre ce schéma et le schéma global du système Osiris. Cette correspondance entre les schémas est une correspondance *path-to-path* .
- un processus de transformation, qui transforme chaque document de la source qui satisfait un schéma concret en un document satisfaisant le schéma global ; ce processus utilise les correspondances mises en place via le processus de correspondance(PC).
- un processus d'extraction, qui analyse *(parse)* chaque document XML dans la source après transformation et extrait les valeurs atomiques ;
- un processus de classement, qui classe les objets après leur extraction ;
- un processus de stockage qui utilise l'entrepôt de données afin de stocker les données qui sont extraites des sources et traitées pour être conformes au schéma global abstrait .
- un processus de traitement des requêtes qui utilise les informations des sous-domaines stables (SDSs) pour obtenir les ISDs satisfaisant la requête.
- un processus d'interrogation qui cherche dans les ISDs les objets satisfaisant la requête et extrait les données pertinentes à la requête.

Dans la section suivante, on explique en détail chaque phase de ce système d'intégration.

## 5.3 Les différentes étapes de l'intégration

### 5.3.1 Représentation du type d'un P-type en XML

Afin de profiter de l'indexation proposée par le système Osiris, le schéma global du système d'intégration doit être un schéma Osiris. Pour cette raison, avant d'étendre le système Osiris pour prendre en compte les documents XML, on a étudié ce modèle afin de déterminer s'il est adapté à l'intégration de documents XML.

Le concept de vue, qui est central pour le modèle des P-types, se révèle être un concept clé pour l'intégration de documents XML. En effet, avec le concept de vue, il n'est pas nécessaire qu'un schéma XML corresponde à un P-type entier, mais seulement un sous-ensemble de ses vues. La vue minimale (cf. chapitre 4) doit nécessairement faire partie de ce sous-ensemble. Une conséquence immédiate est que des schémas très différents peuvent être en correspondance avec le même P-type. En conséquence, l'usage du concept de P-type permet une gestion plus facile des schémas hétérogènes.

## 5.4 Les différentes étapes de l'intégration

Comme le système Osiris est basé sur la notion de P-type, la première question à laquelle nous avons dû répondre est : Est- il possible de transmettre (transformer) un P-type en Osiris dans un schéma XML et vice versa ? Afin de répondre à cette question on a considéré le type du P-type (Personne) dans la figure 5.2 et on a utilisé la recommandation de schéma XML du W3C pour le représenter en XML.

La conclusion est qu'on peut représenter le type d'un P-type en XML.
Dans la suite On montre le type du P-type PERSONNE (décrit dans le chapitre 4) et le schéma correspondant (cf. 5.2) :
**Le type du P-type Personne :**
Attr :

Nom : STRING ;
Sexe : CHARACTER ;
Enfants : setof Personne ;
Age : INT ;
Ad_pers : ADRESSE ;
Service_Militaire : STRING ;
Salaire_mensuel : REAL calc ;
Voitures : setof ref VOITURE ;
Etudes : setof STRING or null ;
Annee : INT or null ;
Statut : STRING or null ;
Diplôme : setof STRING or null ;

Assertions

Sexe in $\{f, m\}$ ;
$0 \leq$ Age $\leq 120$ ;
Service_Militaire in $\{oui, non, exempte, sursitaire\}$ ;

End ;

Comme on le montre dans la figure 5.2, la norme XML schéma donne la possibilité de représenter les différents types de données : les données de type simple (string, integer, float,etc.) et les données de type complexe (TypeAdresse : rue, code_postal, ville). XML schéma donne aussi la possibilité de déterminer la fréquence d'un élément dans le document XML correspondant au schéma. Par exemple, dans la figure 5.2 les éléments de la vue minimale de notre exemple (*nom, sexe, age, salaire, Ad-person*) sont des éléments obligatoires (*not null*) dans le document ; ils doivent apparaître une seule fois et ils sont représentés avec des lignes en gras. Par contre, les éléments représentés par des lignes pointillées sont des éléments optionnels, il peuvent apparaître 0, 1, ou plusieurs fois dans le document ; par exemple, les éléments *études, année, statut, diplôme* sont optionnels.

XML schéma donne aussi la possibilité qu'un élément fasse référence à un autre élément. Par exemple, l'attribut "ref" de type IDREFS de l'élément "enfant" fait référence à l'attribut "ident" de type ID de l'élément "PERSONNE".

De plus XML Schéma offre la possibilité d'utiliser les contraintes d'identité (key et keyref) qui ressemblent à la notion des clés primaires et étrangères dans le modèle relationnel. Dans notre exemple, l'élément "ref-voiture" dans l'élément "PERSONNE" peut référencer un ou plusieurs éléments "id" dans l'élément "VOITURE".

On peut aussi exprimer en XML schéma des assertions sur les valeurs atomiques d'un élément. Par exemple, on peut définir l'assertion définie sur l'élément "âge" du P-type PERSONNE : $0 \leq Age \leq 120$ comme la suite, voir l'annexe(ref) :

```
<xs:element name="age">
  <xs:simpleType>
    <xs:restriction base="xs:integer">
       <xs:minInclusive value="0"/>
       <xs:maxInclusive value="120"/>
       <xs:pattern      value=""/>
    </xs:restriction>
  </xs:simpleType>
</xs:element>
```

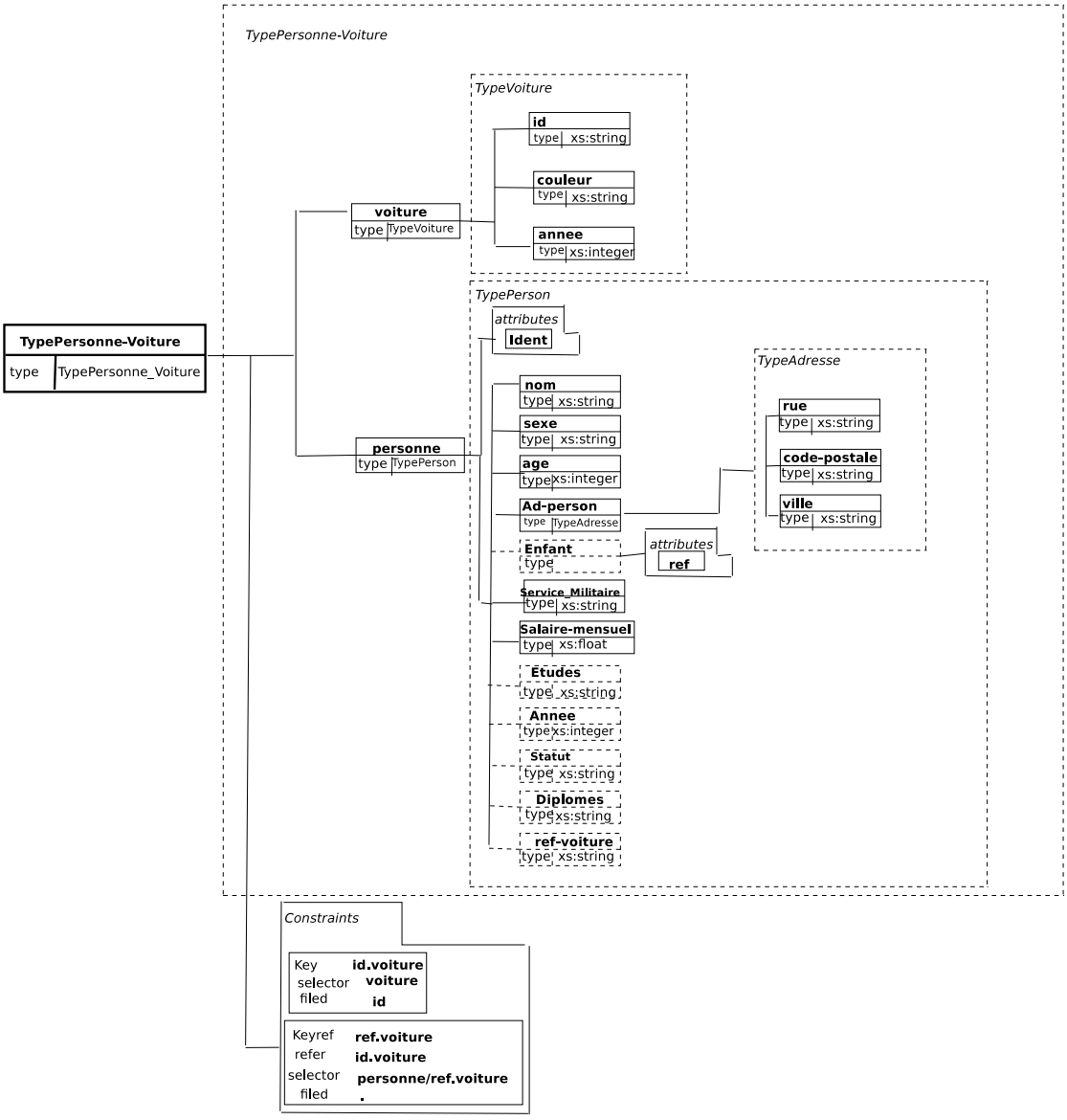


FIG. 5.2 – schéma XML du type Personne

Le code suivant exprime une assertion sur l'élément "sexe" du P-type PERSONNE : $Sexe in \{"f","m"\}$

```
<xs:element name="sexe">
   <xs:simpleType>
     <xs:restriction base="xs:string">
        <xs:enumeration value="f"/>
        <xs:enumeration value="m"/>
     </xs:restriction>
  </xs:simpleType>
</xs:element>
```

De la même façon, on peut définir l'assertion sur l'élément Service-Militaire, Etudes, Statut et Diplôme (voir l'annexe ).

Cet exemple nous a montré la possibilité de représenter le schéma global d'Osiris en utilisant un schéma XMl.

Dans la section suivante nous détaillons la deuxième étape du processus d'intégration basé sur Osiris. Le rôle de cette étape est d'établir la correspondance entre le schéma global d'Osiris et les schémas XML concrets dans la base source.

### 5.4.1 Processus de correspondance

On suppose que la base source de données contient des documents XML qui représentent des données d'un domaine spécifique, que chaque groupe de documents satisfait un schéma XML, qu'on appelle un schéma XML concret ($schema_c$). Ainsi pour une source de données on a un ensemble de schémas XML concrets qui ont des structures différentes, comme le montre la figure 5.3.

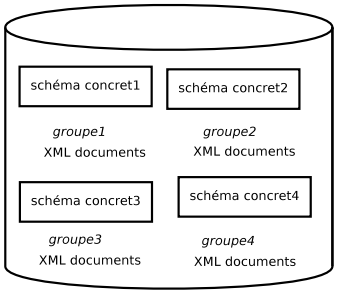


FIG. 5.3 – Une source de données XML

Dans cette étape, on met en correspondance le schéma global d'Osiris représenté en XML $schema_g$ (le schéma obtenu après la phase précédente) et les schémas XML concrets de la base source ($schema_c$ ). On établit ensuite la correspondance (*mapping*) entre le $schema_g$ et chaque $schema_c$ représentant un groupe de documents XML de la base source (voir 5.4)

On considère le schéma global d'Osiris dans notre exemple et on suppose qu'on a le schéma concret de la figure 5.5 dans la base source : On utilise l'approche *path-to-path* pour établir la correspondance entre le schéma concret et le schéma global abstrait.

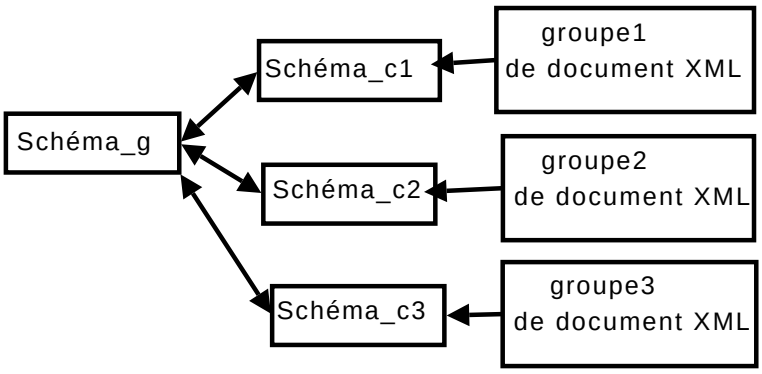


FIG. 5.4 – correspondance entre schémas

| **Eléments du schéma global d'Osiris** | **Eléments de schéma concret** |
|---|---|
| 1. TypePersonne-Voiture/Personne/Nom | Individus/individu/Etat-Civile/Nom-Famille |
| 2. TypePersonne-Voiture/Personne/Age | Individus/individu/Date-Naissance |
| 3. TypePersonne-Voiture/Personne/Diplomes | Individus/individu/Diplomes/Nom-Diplomes |
| 4. TypePersonne-Voiture/Personne/Ref-Voiture | Individus/Individu/Véhicule/Num-Véhicule |
| 5. TypePersonne-Voiture/Personne/Ad-Person/rue<br>TypePersonne-Voiture/Personne/Ad-Person/code-postal<br>TypePersonne-Voiture/Personne/Ad-Person/ville | Individus/individu/Adresse |

TAB. 5.1 – Comparaison entre les éléments du schéma Osiris et les éléments du schéma concret

On décrit dans le tableau 5.1 des exemples de la correspondance entre les éléments du schéma global d'Osiris et les éléments du schéma XML concret de notre exemple : dans l'exemple ci-dessus, on distingue cinq catégories de correspondances. Un travail a été déjà fait au sein de l'équipe pour étudier la correspondance entre des attributs Osiris et des attributs relationnels [Gay, 1999] :

- La correspondance **1** est une correspondance de type atomique ; on n'a pas besoin de faire des calculs pour effectuer cette correspondance, c'est le mapping d'un seul attribut à un autre attribut unique (attribut-attribut).
- La correspondance **2** est une correspondance de type calculé. Dans ce cas, des fonctions de calcul doivent être définies. Par exemple, on doit définir la fonction qui calcule l'âge à partir de la Date-naissance ;
- La correspondance **3** est une correspondance de type collection. Par exemple, une personne peut avoir plusieurs diplômes (individu/Diplomes/Master, individus/Diplomes/doctorat et individus/diplomes/MBA) ;
- La correspondance **4** est une correspondance de type référence.
  Dans notre exemple, on doit mettre en correspondance les deux éléments suivants :

  1. Dans le schéma local, l'élément num-véhicule contenu dans l'élément Individu/véhicule, qui est une référence à l'élément num contenu dans l'élément véhicule.
  2. Dans le schéma global, l'élément ref-voiture contenu dans l'élément Personne qui est une référence à l'élément id contenu dans l'élément voiture.

- La correspondance **5** est une correspondance de type structure. Dans ce cas, plusieurs

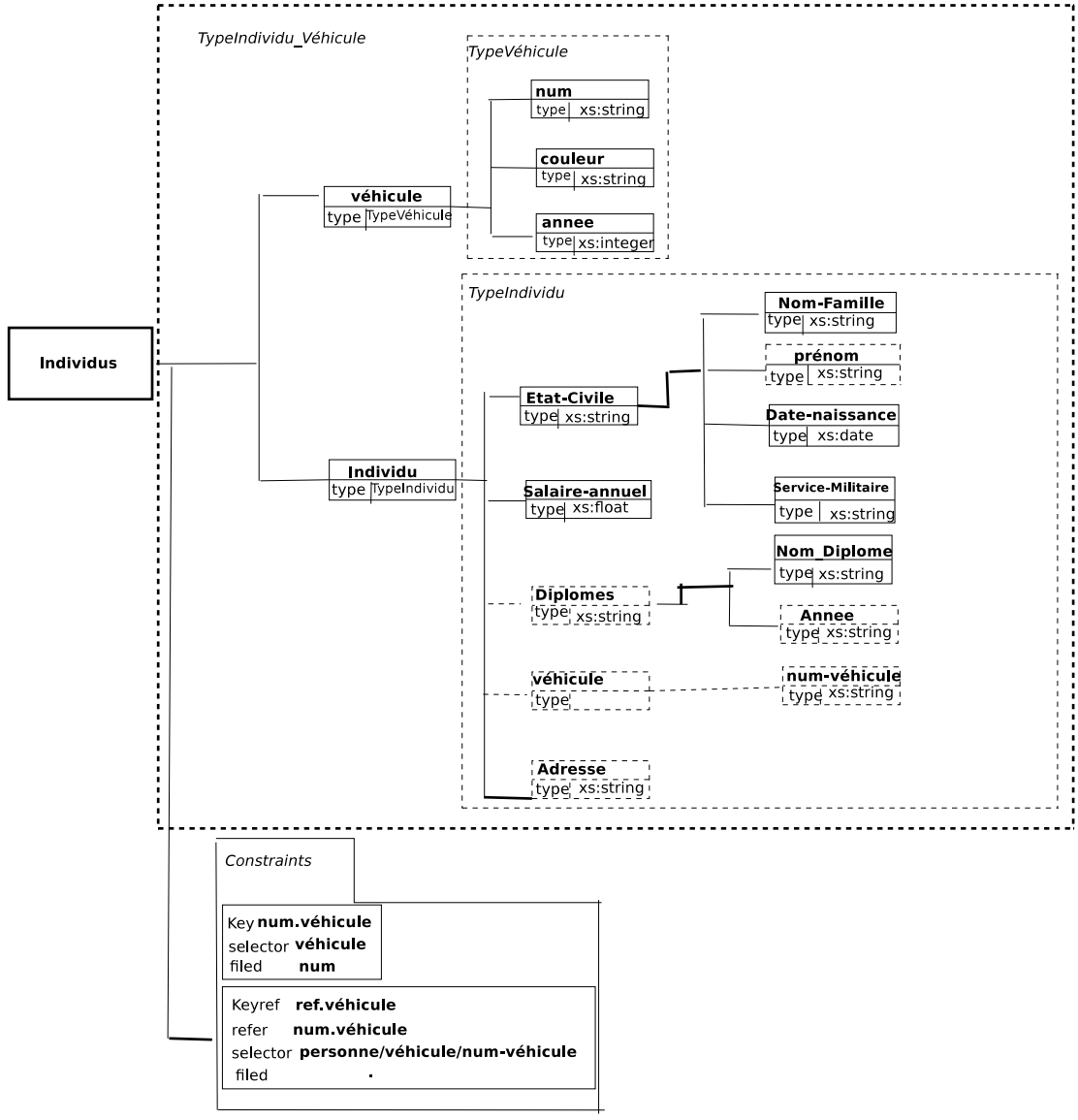


FIG. 5.5 – Le schéma XML concret

éléments peuvent être en correspondance avec un seul élément. Pour effectuer cette correspondance dans notre exemple on définit une fonction de concaténation pour calculer l'adresse comme une composition de rue, code postal et ville.

### 5.4.2 Processus de transformation

Afin de pouvoir profiter de l'indexation offerte par Osiris, le rôle de cette phase est de transmettre (transformer) chaque document XML concret satisfaisant un schéma concret en un document satisfaisant le schéma global d'Osiris. Cette transformation est faite d'une manière semi-automatique en utilisant les informations de correspondance (*mapping*) générées pendant la phase précédente.

Supposons qu'on a **m** groupes de documents XML : $groupe_1$, $groupe_2$,....., $groupe_m$ satisfaisant les schémas concrets $schema_{c1}$, $schema_{c2}$, ...., $schema_{cm}$ respectivement ; on a le schéma global $schema_g$ et les mappings $map_1$, entre $schema_{c1}$ et $schema_g$ et $map_2$, entre $schema_{c2}$ et $schema_g$, ....., $map_m$, entre $schema_{cm}$ et $schema_g$. Chaque document XML dans la base source appartient à un groupe "$groupe_i$" : tel que $1 \leq i \leq m$ et satisfaisant le schéma concret $schema_{ci}$ , le processus de transformation transforme ce document en un document satisfaisant le schéma $schema_g$ en utilisant les information de mapping $map_i$ ; $1 \leq i \leq m$ :

Revenant à notre exemple, on suppose qu'on a dans la base source le document XML suivant qui satisfait, le schéma concret 5.5

```
<Individus>
  <Véhicule>
     <num>60759</num>
     <couleur>Blanche</couleur>
     <annee>2005</annee>
  </Véhicule>
  <Véhicule>
     <num>9807</num>
     <couleur>noir</couleur>
     <annee>2009</annee>
  </Véhicule>

  <Individu>
    <Nom-Famille>Dupont</Nom-Famille>
    <Date-naissance>1980</Date-naissance>
    <Salaire-annuel>40000</Salaire-annuel>
    <Véhicule>
       <num-véhicule>60759</num-véhicule>
    <Véhicule>
 </Individu>
</Individus>
```

La phase de transformation va transformer ce document en un document satisfaisant le schéma global représenté dans la figure 5.2 :

```
<TypePersonne_Voiture>
  <voiture>
```

```
     <id>60759</id>
     <color>Blanche</color>
     <year>2005</year>
  </voiture>
  <voiture>
     <id>9807</id>
     <color>noir</color>
     <year>2009</annee>
 </voiture>

  <person>
    <Nom-Famil>Dupont</Nom-Famille>
    <age>28</age>
    <Salaire-mensuel>333.33</Salaire-annuel>
    <Véhicule>
       <num-véhicule>60759</num-véhicule>
    <Véhicule>
 </person>
</TypePersonne_Voiture>
```

### 5.4.3 Processus d'extraction

Suite à la phase précédente, tous les documents de la base source sont transformés en des documents satisfaisant le schéma global en Osiris. Dans la phase d'extraction on parcourt (*parse*) l'arbre DOM de chaque document afin de récupérer l'ensemble de feuilles qui représente le maximum d'attributs (voir figure 5.6).

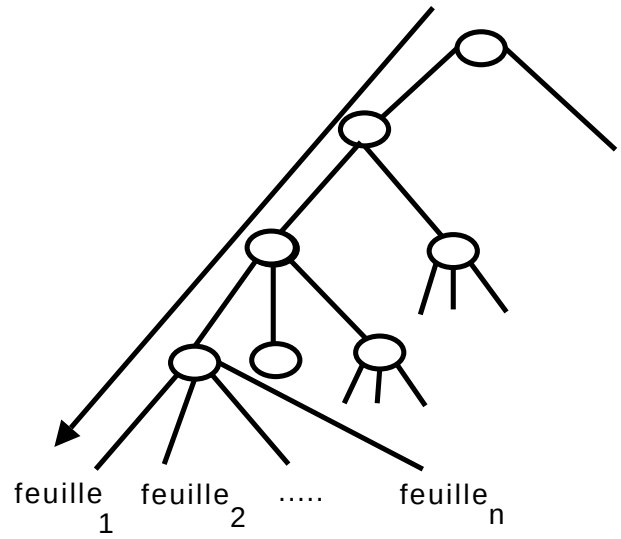


Fig. 5.6 – Parcours de l'arbre DOM d'un document XML

### 5.4.4 Processus de classement

Le système Osiris effectue le classement d'objets comme expliqué dans le chapitre précédent. Ce processus utilise les informations des Sous-Domaines Stables (SDSs) et envoie les résultats de classement à la structure d'indexation (ISD). On doit refaire ce processus à chaque mise à jour du schéma global (en cas de modifications du schéma ou de changement des assertions).

En Osiris, classer un objet dans un P-type revient à classer son Eq-classe. Les étapes du classement sont les suivantes :

1. pour chaque attribut valué, on détermine le sous-domaine stable correspondant.
2. on déduit l'Eq-classe de l'objet.
3. on classe le n-uplet ainsi obtenu en utilisant le réseau de classement.

### 5.4.5 Processus de stockage (entrepôt de données)

On stoke les données obtenues après la phase d'extraction dans une table relationnelle (entrepôt de données) qui a le schéma suivant :
Table_de_Stockage_de_données(oid, docID, $att_1$, $att_2$, ..., $att_m$) :

- OID est l'identificateur de l'objet dans le système Osiris.
- docID est l'identificateur du document qui contient la donnée.
- $att_1$, $att_2$, ..., $att_m$ sont les attributs classificateurs d'Osiris et tous les autres attributs représentant l'ensemble de feuilles de l'arbre DOM de chaque document XML dans la source.

### 5.4.6 Traitement des requêtes

Dans le système Osiris, l'utilisateur pose sa requête sur le schéma global en Osiris. Le système utilise les informations de SDSs pour évaluer la requête en suivant les étapes suivantes [Simonet, 1996]

- le système réécrit la requête en termes de sous-domaines stables et détermine les ISDs potentiels correspondant à la requête.
- parmis ces ISDs, le système détermine les ISDs dont les objets valident la requête de façon sûre (ISDs valides), et ceux pour lesquels un teste supplémentaire est nécessaire (ISDs potentiels).

Suite à ces deux étapes, on détermine les ISDs effectifs de la requête, c'est à dire ceux stockés dans l'espace ISD du système car ils disposent d'au moins un objet stocké. Les ISDs valides donnent les objets valides pour la requête, tandis que les ISDs potentiels donnent les objets pour lesquels un test supplémentaire doit être effectué afin de déterminer s'il est une réponse à la requête. Pour les ISDs potentiels on définit les conditions à vérifier pour qu'un objet de l'ISD satisfasse la requête.

### 5.4.7 Processus d'interrogation

Dans l'ISD de notre système, pour chaque objet on a une référence vers la donnée source correspondante qui est stockée dans l'entrepôt de données. Pour les objets validant la requête on utilise leur référence vers l'entrepôt afin de pouvoir récupérer les attributs demandés dans la requête. Pour les objets éventuellement valides, il est nécessaire de vérifier que ces objets satisfont les conditions supplémentaires pour confirmer qu'un objet satisfait ou non la requête.

### 5.4.8 Processus de mise à jour d'une donnée d'un document XML

Pour traiter le cas de la mise à jour d'une donnée d'un document XML dans la source, on stocke dans une mémoire temporaire (une table relationnelle) les méta-données pour la mise à jour, qui contiennent la correspondance entre chaque donnée de la source et l'objet correspondant en Osiris. Le schéma de la table relationnelle est de la forme suivante :
Méta_données(id_Source, id_OSIRIS) où :
id_Source est l'identificateur de la donnée dans la source.
id_OSIRIS est l'identificateur de l'objet corresponant en OSIRIS (l'OID de l'objet en OSIRIS).
Quand la valeur d'une donnée d'un document XML dans la source change, la table "Méta_données" fournit l'objet Osiris coresspondant à cette donnée modifiée. On distingue plusieurs cas :

1. S'l y une modification de la valeur d'un atrribut classificateur (par exemple âge) d'une donnée dans la source, on refait le processus de classement pour l'objet Osiris correspondant à cette donnée et on modifie la valeur du n-uplet correspondant dans la table de stockage des données.
2. S'l y une modification de la valeur d'un atrribut non classificateur d'une donnée dans la source, on modifie seulement la valeur de l'attribut dans la table de stockage des données.
3. S'l y une donnée supprimée dans la source, on supprime l'objet OSIRIS correspondant à cette donnée dans la table d'indexation ISD et on supprime le n-uplet correspondant dans la table de stockage des données.
4. S'l y une nouvelle donnée dans la source, on classe l'objet correspondant dans la table d'indexation ISD et on ajoute le n-uplet correspondant dans la table de stockage des données.

## 5.5 Exemple dans le domaine médical

Dans cette section, nous présentons un exemple d'application de notre approche dans le domaine médical. Notre exemple modélise un hôpital qui est composé de plusieurs éléments : Personne, Service, Soin, Séjour, Médicaments, Réaliser, Résultat et Analyses. En utilisant le concept de P-type, qui permet la spécification des points de vue sur un domaine précis, on définit Le P-type PERSONNE, on commençant par la vue minimale PERSONNE, puis on définit les autres vues : MEDECIN, PATIENT et ADMINISTRATEUR. On peut aussi ajouter d' autres vues comme MINEUR, ADULTE, SENIOR, FEMME et HOMME dans le P-type PERSONNE, comme montré dans le figure 5.7.

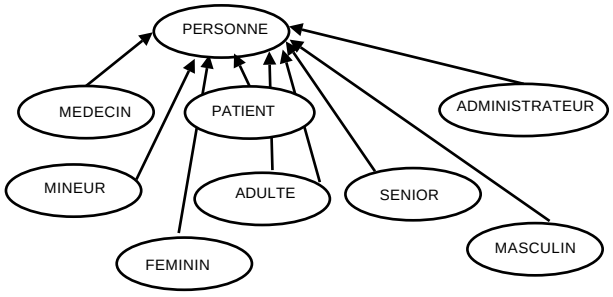


FIG. 5.7 – Le P-type PERSONNE pour l'exemple "hôpital"

Une PERSONNE peut être MEDECIN, PATIENT ou ADMINISTRATEUR, etc.. ou plusieurs d'entre eux à la fois. Une personne donnée est un modèle de la vue minimale et peut appartenir à une ou plusieurs autres vues. Elle peut aussi appartenir seulement à la vue minimale PERSONNE.

Le P-type PERSONNE de notre exemple "hôpital" est défini de la manière suivante :

**view** PERSONNE - - Vue minimale du P-type PERSONNE
**attr** id : STRING ;
nom : STRING ;
sexe : CHAR **in** $\{f, m\}$ ; const ; - - assertion a1
age : INT **in** ]0,120] **calc** ; - - assertion a2
**key** id - - Clé externe, base de la contrainte d'unicité
**methods ...** - - Spécification des autres fonctions
**assertions**
Sexe CHAR **in** $\{f, m\}$ ;
Age INT **in** ]0,120] ;
**end** ; - - Un attribut OID : toid.est créé automatiquement dans la vue minimale

**view** MEDECIN : PERSONNE - - MEDECIN spécialise PERSONNE
**a**ttr salaire : REAL ;
service : STRING ; - - Une vue dans une autre P-type
**assertions**
spécialité : setof STRING in {P*édiatrie*, *Cardiologie*, *Radiologie*,n*éphrologie*} ;
**end** MEDECIN ;

**view** PATIENT : PERSONNE - - MEDECIN spécialise PERSONNE
**attr**
maladie : STRING ;
**assertions**
maladie : STRING in {*cardiaque*, *Allergie*, *Anpholoanza*, *Renal*, *Grippe*} ;
**end** PATIENT ;

**view** ADMINISTRATEUR : PERSONNE – DMINISTRATEUR spécialise PERSONNE
**attr**
catégorie : STRING ;
**assertions**
catégorie : STRING in $\{A, B, C, AB, AC\}$ ;
**end** ADMINISTRATEUR ;

On peut aussi définir les vues suivantes :
view ADULT : PERSONNE assertions age $\geq$ 18 end ;
view MINEUR : PERSONNE assertions age $<$ 18 end ;
view SENIOR : PERSONNE assertions age $\geq$ 65 end ;
view FEMME : PERSONNE assertions sexe = f end ;
view HOMME : PERSONNE assertions sexe = m end ;

Le premier processus est la description du type du P-type PERSONNE et ses relations avec les autres concepts (Service, Soin, Séjour, Médicaments, réalise_analyse et résultats) en utilisant

la recommandation de schéma XML du W3C (voir figure 5.8) ; cette figure représente le schéma global de notre système d'intégration.

On note qu'on utilise les contraintes d'identité (key et keyref) pour faire la référence d'un élément vers un autre élément.
Par exemple dans la contrainte :

```
key id_service
selector Personnes
field code_service

keyref ref_service
refer id_service
selector Service/numserv
field .
```

On a défini que l'élément *code_ service* dans Personnes est une référence vers l'élément *numserv* dans Service. On a aussi défini que l'élément *chef* dans Service est une référence vers l'élément *id* dans Personnes, en utilisant la contrainte :

```
key id_chef
selector Service
field  chef

keyref ref_medcin
refer id_chef
selector Personnes/id
field .
```

De la même manière, on a défini les autres références :

```
numpat dans Soin est une référence vers id dans Personnes.
nummed dans Soin est une référence vers id dans Personnes.
refmed dans Soin est une référence vers refmed dans médicaments.
numpat dans Séjours est une référence vers id dans Personnes.
```

et ainsi de suite.

On suppose qu'on a un groupe de documents dans la base source satisfaisant le schéma concret ( $schema_c$) de la figure 5.9. Le schéma décrit le secteur des urgences dans un hôpital (SAMU) qui est composé de plusieurs éléments : Individus, Secteur, Traitement et Soin.

On utilise l'outil ALTOVA [13] pour faire la correspondance entre le $schema_g$ et le $schema_c$, comme le montre la figure 5.10, le *mapping* est de type *path-to-path* : on montre dans le tableau 5.2 toutes les correspondances extraites à partir de l'exemple "Hôpital".
On définit des fonctions afin de réaliser des correspondances plus complexes de type calculé.

Par exemple, le code XSLT qui permet de définir la correspondance entre la date de naissance et l'âge , et qui permet donc de calculer l'âge, est le suivant :

```
<âge>
  <xsl:value-of select="(number(string('2008'))-number((string(cv/date_naissance)))"/>
</âge>
```

[13] ALTOVA, *http ://www.altova.com/*, XML Editor, Data Management, UML and Web Service tools.

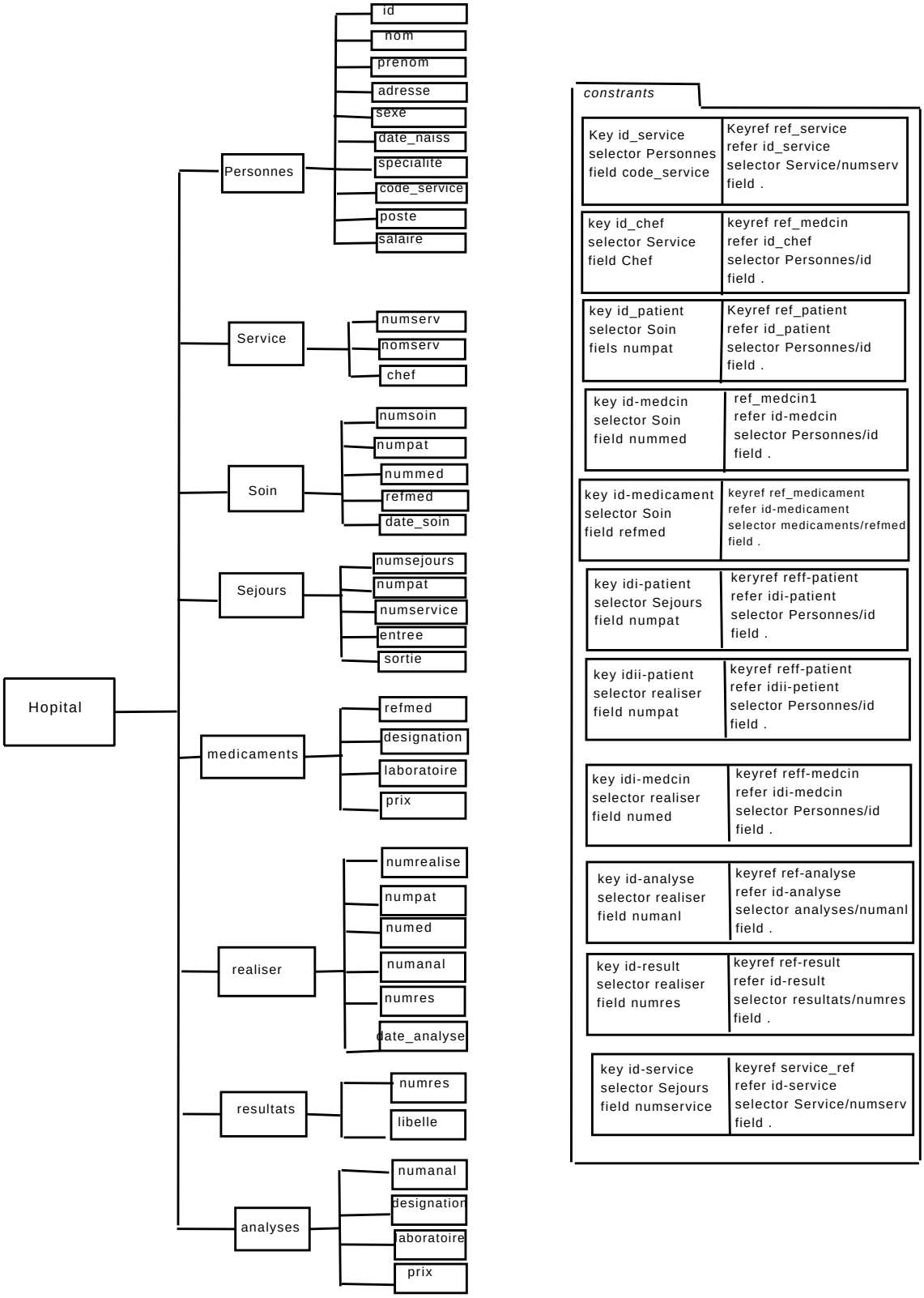


FIG. 5.8 – schéma XML global d'un hôpital

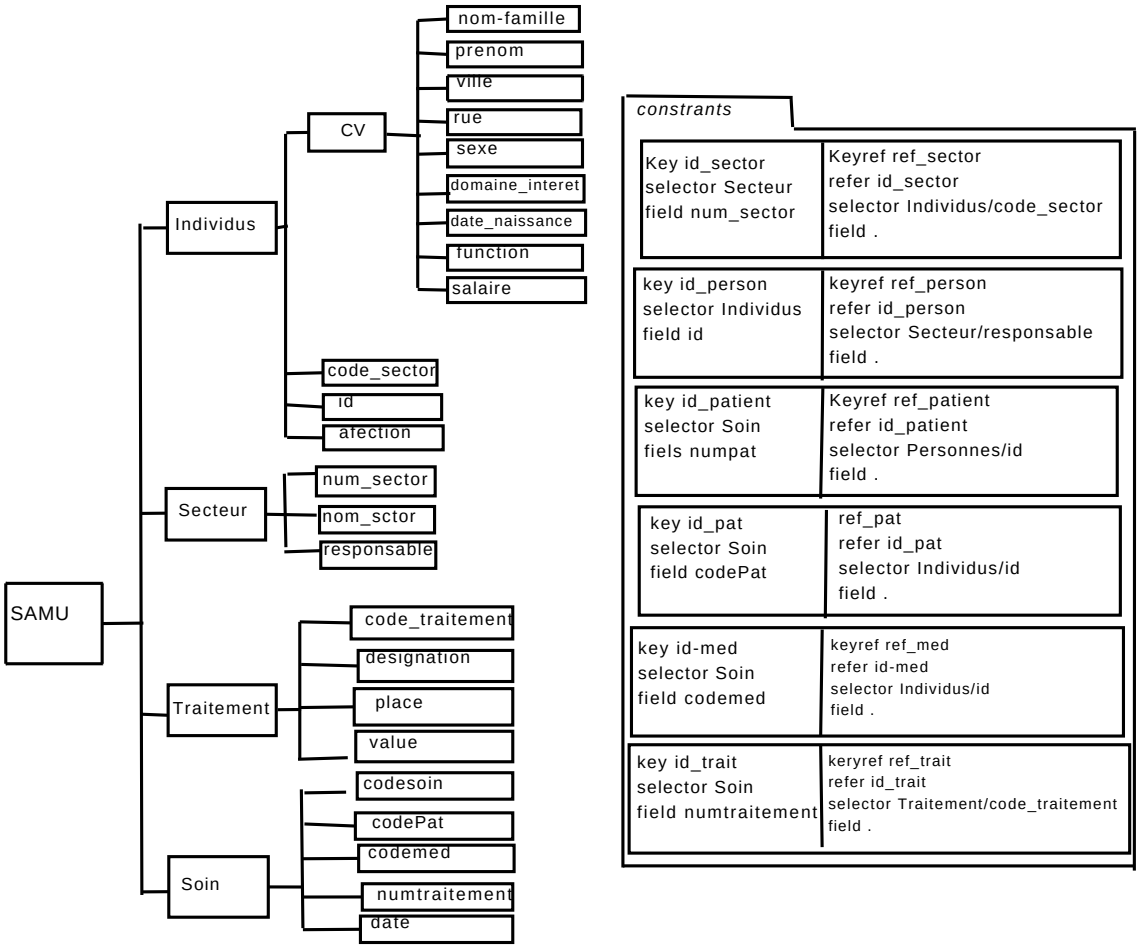


FIG. 5.9 – schéma XML local

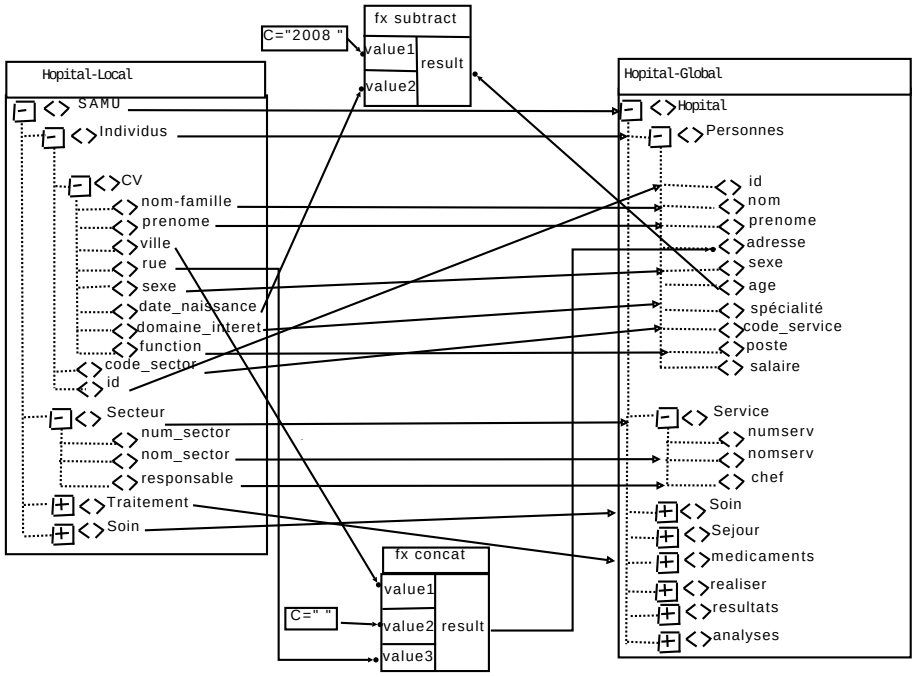


FIG. 5.10 – mapping entre le schéma XML local et le schéma XML global d'un hôpital

Autre exemple : on définit une fonction (*concat*) pour calculer l'adresse comme la concaténation de la ville et de la rue avec le code XSLT suivant :

```
<adresse>
     <xsl: value-of select="concat(concat(string(cv/ville),''),string(cv/rue))"/>
</adresse>
```

Après avoir défini le mapping entre le schéma global et le schéma local en utilisant XSLT, on passe à la transformation.
Supposons qu'on a les documents XML d1, d2 et d3 dans la base source (voir Annexe B). Ces documents satisfont le schéma local de la figure 5.9. Comme il a été expliqué dans la section précédente, le processus de transformation transforme ces documents en des documents qui satisfont le schéma global. Dans le chapitre suivant, nous expliquons notre algorithme de transformation pour un réperatoire qui contient plusieurs documents XML.

Dans la phase d'extraction, on parcourt l'arbre DOM de chaque document après sa tranformation afin d'extraire les feuilles de l'arbre (valeurs atomiques).
Après l'extraction des feuilles, le système Osiris effectue le classement d'objets.

Pour notre application, on considère les assertions sur les attributs *sexe* et *âge* :

Sexe : CHAR in $\{f, m\}$ ;

Age : INT in $]0, 120]$.

On définit les Sous-Domaines Stables :
Sexe : $d_{11} = \{f\}$, $d_{12} = \{m\}$
Age : $d_{21}$=[0,18[ , $d_{22}$=[18,35[, $d_{22}$=[35,120]
Les vues dans notre exemple sont PERSONNE, PATIENT, MEDECIN, ADMINISTRATEUR ADULTE, MINEUR, SENIOR, FEMME et HOMME. Osiris fournit la table d'indexation suivante (voir 5.11) avec les objets indexés dans cette table :

| **Eléments du schéma local** | **Eléments de schéma global** |
|---|---|
| SAMU | Hopital |
| SAMU/Individus | Hopital/Personnes |
| SAMU/Individus/CV/nom-famille | Hopital/Personnes/nom |
| SAMU/Individus/CV/prenom | Hopital/Personnes/prenom |
| SAMU/Individus/CV/ville<br>SAMU/Individus/CV/rue | Hopital/Personnes/adresse |
| SAMU/Individus/CV/date-naissance | Hopital/Personnes/age |
| SAMU/Individus/CV/domaine-interet | Hopital/Personnes/spécialité |
| SAMU/Individus/CV/function | Hopital/Personnes/poste |
| SAMU/Individus/code-sector | Hopital/Personnes/code-service |
| SAMU/Individus/id | Hopital/Personnes/id |
| SAMU/Secteur | Hopital/Service |
| SAMU/Secteur/num-sector | Hopital/Service/numserv |
| SAMU/Secteur/nom-sector | Hopital/Service/nomserv |
| SAMU/Secteur/responsable | Hopital/Service/chef |
| SAMU/Traitement | Hopital/medicaments |
| SAMU/Traitement/code-traitement | Hopital/medicaments/refmed |
| SAMU/Traitement/designation | Hopital/medicaments/designation |
| SAMU/Traitement/place | Hopital/medicaments/laboratoire |
| SAMU/Traitement/prix | Hopital/medicaments/value |
| SAMU/Soin/codesoin | Hopital/Soin/numsoin |
| SAMU/Soin/codePat | Hopital/Soin/numpt |
| SAMU/Soin/codemed | Hopital/Soin/nummed |
| SAMU/Soin/numtraitement | Hopital/Soin/refmed |
| SAMU/Soin/date | Hopital/Soin/date-soin |

TAB. 5.2 – Correspondance entre les éléments du schéma local et les éléments du schéma global

on stocke les objets dans une table relationnelle avec tous les attributs possibles (voir 5.3, 5.4, 5.5).

Prenons des exemples de requête dans l'application "Hôpital" :

**Requête1 :** Afficher les noms et les prénoms des PATIENT de sexe féminins et âgés de plus de 25 ans :

Requête1 :(PATIENT | Age>25 et Sexe= f) [nom, prénom]

On écrit la requête en utilisant les Sous-Domaines Stables :
la figure 5.12 représente les SDSs d'intérêt et leurs valeurs :

On détermine les Eq-classes correspondantes et leurs valeurs :

($d_{11}$,$d_{22}$, P), ($d_{11}$,$d_{23}$, V)

On utilise la table d'indexation (voir 5.11) pour déterminer les objets qui appartiennent à ces Eq-classes ; on obtient :

OID2, OID4 qui appartiennent à ($d_{11}$,$d_{22}$)

| Vues et SDSs / Eq-classe | PERSONNE | MEDECIN | PAIENT | ADMINSTRATEUR | FEMME | HOMME | MINEUR | ADULTE | SENIOR | [0,18[ | [18,35[ | [35,120] | "F" | "M" | Les OID d'objets |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| [d11,d21] | V | I | V | I | V | I | V | I | I | V | I | I | V | I | OID8 |
| [d11,d22] | V | I | V | I | V | I | I | V | I | I | V | I | V | I | OID2 |
| [d11,d23] | V | V | I | I | V | I | I | V | I | I | I | V | V | I | OID7 |
| [d12,d21] | V | I | V | I | I | V | V | I | I | V | I | I | I | V | OID1 |
| [d12,d22] | V | I | V | I | I | V | I | V | I | I | V | I | I | V | OID6 |
| [d12,d22] | V | V | I | I | I | V | I | V | I | I | V | I | I | V | OID3 |
| [d11,d22] | V | V | I | I | V | I | I | V | I | I | V | I | V | I | OID4 |
| [d12,d23] | V | V | I | I | I | V | I | V | V | I | I | V | I | V | OID5 |

FIG. 5.11 – Table d'indexation d'objets de l'exemple "Hôpital"

| OID | Id_DOC | nom | prenom | Adresse | sexe | age | code_service |
|---|---|---|---|---|---|---|---|
| OID1 | d3 | AHMAD | Yehia | Grenoble Alsace Lorraine | M | 1 | S4 |
| OID2 | d2 | ADAM | Janette | Grenoble Bon Pasteur | F | 34 | S3 |
| OID3 | d3 | MICHEL | Dimitry | Grenoble Grand Sablon | M | 30 | S3 |
| OID4 | d2 | BERNARD | Maryana | Grenoble Victor Hugo | F | 30 | S2 |
| OID5 | d1 | ABDALLAH | Bassam | Paris Jeanne d'Arc | M | 65 | S1 |
| OID6 | d1 | ALI | Omar | Paris Jean Jaures | M | 27 | S1 |
| OID7 | d2 | MORIERA | Janette | Grenoble Fleurs | F | 50 | S3 |
| OID8 | d3 | MARCHALLE | Sylvie | Grenoble Peupliers | F | 3 | S3 |

TAB. 5.3 – stockage des données

OID7 qui appartient à ($d_{11}$,$d_{23}$)

Parmi ces objets on extrait ceux qui valident la vue PATIENT : on obtient l'objet OID2 comme réponse potentielle à la requête. On parcourt ensuite la table de stockage de données pour extraire le nom et le prénom de cet objet (OID2). La réponse à cette requête est :

<nom : ADAM, prénom : Janette>.

**Requête2 :** Afficher le nom et la spécialité des Médecins de sexe masculin agés de plus de 30 ans :

Requête2 : (MEDECIN | Age>40 et Sexe= M) [nom, spécialité]

On écrit la requête en utilisant les Sous-Domaines Stables :

la figure 5.13 représente les SDSs d'intérêt et leurs valeurs :

On détermine les Eq-classes correspondantes et leurs valeurs :

($d_{12}$,$d_{23}$, V)

On utilise la table d'indexation (voir 5.11) pour déterminer les objets qui appartiennent à cette Eq-classe : on obtient l'objet OID5 qui valide la vue MEDECIN.

On utilise la table de stockage de données pour extraire le nom et la spécialité de l'objet OID5. La réponse à cette requête est :

<nom : ABDALLAH, Spécialité : Cardio>.

**Requête3 :** Afficher les noms et les adresses des personnes (patient ou médecin) de sexe féminin :

Requête3 : (PATIENT OR MEDECIN | Sexe= f) [nom, adresse]

| **maladie** | **numserv** | **nomserv** | **chef** | **spécialité** | **numsoin** | **numpat** | **nummed** |
|---|---|---|---|---|---|---|---|
| Anpholoanza | S4 | néphrologues | OID3 | NULL | NULL | NULL | NULL |
| Alergie | S3 | Pediatrie | OID7 | NULL | NULL | NULL | NULL |
| NULL | S4 | néphrologues | OID3 | néphrologie | NULL | NULL | NULL |
| NULL | S2 | radiologie | OID4 | radiologie | NULL | NULL | NULL |
| NULL | S1 | cardiologie | OID5 | cardio | soi1 | OID6 | OID5 |
| Cardique | S1 | cardiologie | OID5 | NULL | soi1 | OID6 | OID5 |
| NULL | S3 | Pediatrie | OID7 | pédiatre | NULL | NULL | |
| Gripe | S4 | néphrologues | OID3 | NULL | NULL | NULL | NULL |

TAB. 5.4 – stockage des données

Grenoble Peupliers

| **refmed** | **date_soin** | **refmedi** | **designation** | **laboratoire** | **prix** |
|---|---|---|---|---|---|
| NULL | NULL | NULL | NULL | NULL | NULL |
| NULL | NULL | NULL | NULL | NULL | NULL |
| NULL | NULL | NULL | NULL | NULL | NULL |
| NULL | NULL | NULL | NULL | NULL | NULL |
| T1 | 2008 | T1 | Med1 | Michalon | 260 |
| T1 | 2008 | T1 | Med1 | Michalon | 260 |
| NULL | NULL | NULL | NULL | NULL | NULL |
| NULL | NULL | NULL | NULL | NULL | NULL |

TAB. 5.5 – stockage des données

| **nom** | **adresse** |
|---|---|
| MARCHALLE | Grenoble PEUPLIERS |
| ADAM | Grenoble BON PASTEUR |
| BERNARD | Grenoble VICTOR HUGO |
| MORIERA | Grenoble FLEURS |

TAB. 5.6 – la réponse à la requête3

On écrit la requête en utilisant les Sous-Domaines Stables :
la figure 5.14 représente les SDSs d'intérêt et leurs valeurs :

On détermine les Eq-classes correspondantes et leurs valeurs :
($d_{11}$,$d_{21}$, V), ($d_{11}$,$d_{22}$, V), , ($d_{11}$,$d_{23}$, V)

On utilise la table d'indexation (voir 5.11) pour déterminer les objets qui appartiennent à ces Eq-classes, ce qui donne :

OID8 qui appartient à ($d_{11}$,$d_{21}$), il valide la vue PATIENT

OID2, OID4 qui appartienent à ($d_{11}$,$d_{22}$), ils valident les vues PATIENTS et MEDECINS

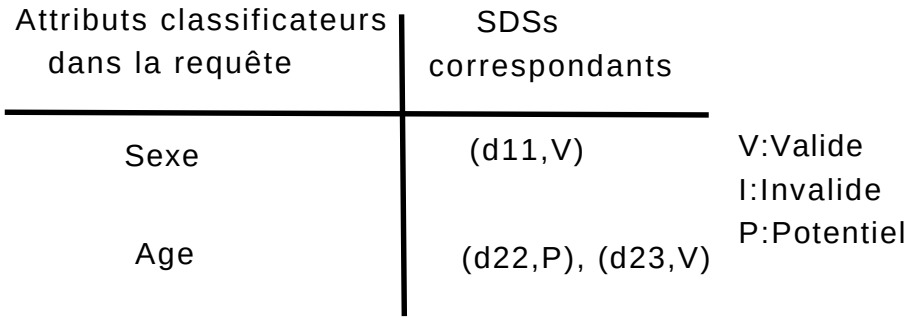


FIG. 5.12 – les SDSs correspondant à la requête

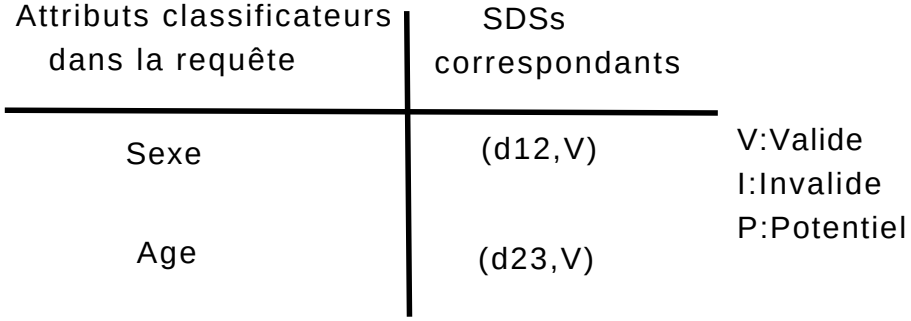


FIG. 5.13 – les SDSs correspondant à la requête

respectivement.

OID7 qui appartient à ($d_{11}$,$d_{23}$), il valide la vue MEDECIN

On parcourt ensuite la table de stockage de données pour extraire les noms et les adresses de ces objets (OID2). La réponse à cette requête est montrée dans la table 5.6 :

## 5.6 Comparaison entre le système OSIX et les autres systèmes

Dans cette section nous présentons une comparaison entre le système OSIX et les systèmes Xylème et VIMIX étudiés dans l'état de l'art (voir chapitre3), qui sont conçus pour l'intégration de données XML en utilisant les vues. La table 5.7 récapitule les caractéristiques des trois approches. Cette comparaison nous a conduits à mieux cerner l'intérêt du système OSIX pour l'optimisation sémantique de requêtes par rapport aux systèmes étudiés. L'omission de la phase de traduction de requêtes optimise le temps de traitement. Par ailleurs, l'utilisation de l'indexation d'Osiris a permis de déterminer automatiquement à quelles vues appartient un document XML. En conséquence, l'espace de recherche pour les documents qui font partie à la requête peut être réduit à l'espace des vues satisfaisant la requête, ce qui constitue l'optimisation sémantique de requêtes.

## 5.7 Conclusion

Dans ce chapitre, on a proposé une approche pour intégrer les documents XML dans un *middleware* sémantique destiné aux bases de données relationnelles et XML. Cette approche

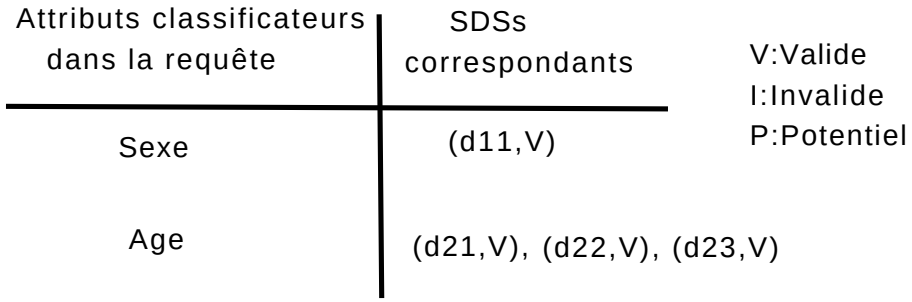


FIG. 5.14 – les SDSs correspondant à la requête

est basée principlement sur trois processus : transformation, extraction et interrogation. L'étape clé de cette approche est la définition de la correspondance entre un schéma global en Osiris et plusieurs schémas XML concrets. Cette étape est liée directement au processus de transformation.

Pour établir cette correspondance on a utilisé une étape intermédiaire qui sert à :

1. définir les types d'un P-type, c'est à dire le type maximal et les types de chacune de ses vues ;
2. traduire les types résultats dans un schéma XML ;

Les schémas issus d'un P-type ne sont pas indépendants mais ils sont liés par des relations hiérarchiques. En conséquence, pour chaque schéma XML concret, il est possible de calculer la correspondance entre les éléments du schéma XML correspondant au type maximal et les éléments d'un schéma XML concret. On a utilisé la correspondance de type path-to-path, qui s'est avérée fournir les meilleurs résultats. Pendant la phase de mise en place de la correspondance, une procédure peut être attachée afin d'effectuer des calculs en cas de besoin, par exemple pour calculer la valeur de l'âge à partir de la valeur de la date de naissance.

On a aussi montré que notre système utilise la structure d'indexation (IDSs) offerte par Osiris afin d'effectuer l'optimisation sémantique des requêtes. Cela permet au système de déterminer automatiquement à quelles vues appartient un document XML. En conséquence, l'espace de recherche pour les documents qui font partie de la réponse à la requête peut être réduite à l'espace des vues satisfaisant la requête. Cette optimisation est importante parce que l'exploitation actuelle d'un grand nombre de documents reste un problème, même si le principe d'utilisation d'un schéma global abstrait constitue un progrès important pour l'exploitation des documents XML d'une manière homogène.

Dans ce chapitre on a justifié notre reflexion par un exemple détaillé dans le domaine médical. Le chapitre suivant présente une implémentation de l'approche OSIX définissant l'outil relié. Le choix de l'utilisation du système Osiris pour notre approche d'intégration de données XML a été justifié par la comparaison du système OSIX avec les systèmes existants VIMIX, Xylème, XyView étudiés dans le chapitre 4.

| | **Xylème** | VIMIX | OSIX |
|---|---|---|---|
| Méthode pour définir les*mappings* | path-to-path | graphe de mapping | path-to-path |
| Approche utilisée pour l'intégration de données | entrepôt de données | entrepôt de données | entrepôt de données |
| Les vues | matérialisées | matérialisées | semi-matérialisées |
| Approche pour le schéma global | GAV et LAV à la fois | hybride | GAV |
| Schéma global | une groupe de DTDs abstraits | un ensemble de vues | une hiérarchie de vues |
| Méthode de stockage de données | stockage des documents XML par cluster | tables relationnelle | tables relationnelles |
| Traduction de requêtes | phase indispensable | phase indispensable | pas nécessaire |
| Langage d'interrogation | XQuery | XQuery | SQL |
| Avantages de l'approche | Optimisation de requêtes | Optimisation du temps de stockage | optimisation sémantique de requêtes |
| Méthode utilsée pour l'optimisation | l'indexation de documents XML et le distribution de vues | sépare le stockage de données et le stockage de mapping | utilisation d'indexation d'Osiris |

Tab. 5.7 – Comparaison entre les différentes approches

# Chapitre 6

# Expérimentation et Évaluation : L'outil OSIXT (OSIX Tool)

## 6.1 Introduction

Dans ce chapitre, nous présentons le processus général que nous avons suivi pour implémenter le système OSIX, conçu pour l'intégration de document XML. Nous présentons l'outil OSIXT( OSIX Tool) qui permet d'interroger une source de données XML. Cet outil permet aussi l'optimisation sémantique des requêtes posées sur une source de documents XML, en utilisant l'indexation proposée par Osiris, qui est basée sur les vues et les SDSs. Nous avons vu dans le chapitre 5 que le système OSIX permet d'effectuer la transformation des documents source satisfaisant des schémas locaux en des documents satisfaisant le schéma global. Ceci est réalisé par un module appelé "Transformation", intégré dans l'outil implémenté.

Le système dispose également d'un module "Extraction" qui permet d'extraire les valeurs atomiques (les feuilles de l'arbre DOM) des documents XML représentant la base source. Notre système fournit aussi un module "Interrogation" qui permet de saisir une requête sur une source de documents XML et donne la réponse appropriée.

Par la suite, nous donnons une description détaillée des trois modules, leurs algorithmes ainsi que leurs interfaces logicielles.

Ce chapitre est organisé comme suit :

- Dans la section 6.2, nous présentons l'architecture générale du système .
- Dans la section 6.3, nous présentons l'interface graphique qui permet de spécifier les trois modules de base de l'outil OSIXT.
- Dans la section 6.4, nous présentons les algorithmes des trois modules (Transformation, Extraction, Interrogation) et leur description.

## 6.2 Présentation générale

### 6.2.1 Architecture fonctionnelle

La figure 6.1 présente l'architecture fonctionnelle de notre système. Le système OSIX est composé de trois interfaces qui permettent de faire la transformation (bouton "convert"), l'extraction (bouton "Export") et l'interrogation (bouton "Query") d'une base source de documents XML.

L'interface utilisateur permet de :

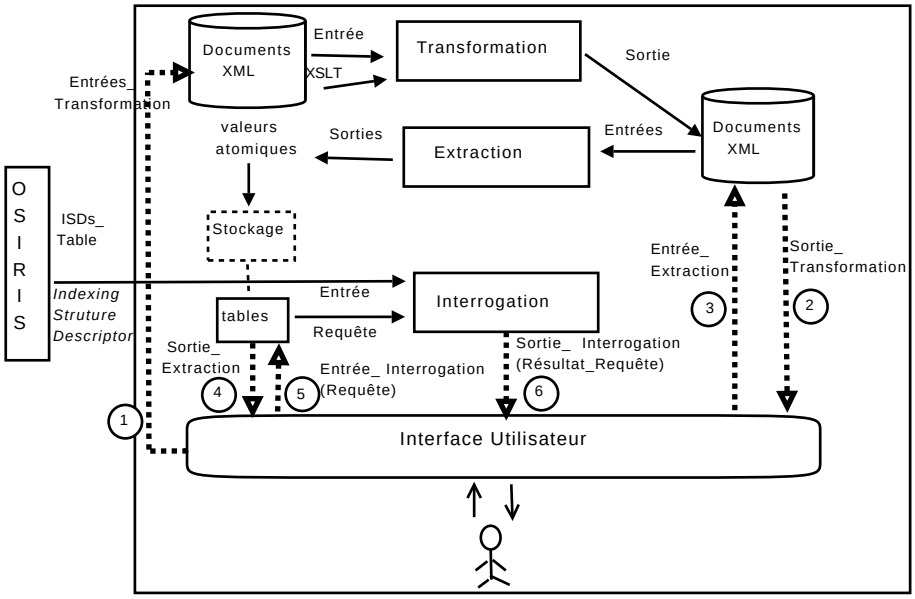


FIG. 6.1 – Architecture fonctionnelle du système

- Entrer un ensemble de documents XML déjà répertoriés dans un dossier qui satisfait un schéma local XML afin de faire la transformation. L'utilisateur aura comme résultat un répertoire de fichiers XML transformés satisfaisant le schéma global.
- Sélectionner un répertoire de fichiers XML transformés pour exécuter le module "Extraction". Le résultat de cette étape est une table contenant les valeurs atomiques de fichiers XML.
- Saisir une requête SQL qui va consulter la base afin d'avoir la réponse appropriée.

Le traitement de requêtes utilise le mécanisme du classement d'Osiris caractérisé par l'utilisation des vues et des sous-domaines stables afin d' optimiser le temps de réponse de la requête.

La table 6.1 représente une description détaillée de l'architecture fonctionnelle du système.

## 6.3 Implémentation

Nous avons implémenté le système OSIX en utilisant le langage Java [Java], dans sa version 6. Le choix du langage Java présente les avantages suivants :

- Langage largement répandu. L'existence de nombreuses librairies décrivant le langage a facilité le développement de l'application.
- Langage orienté-objet.
- Les compilateurs sont gratuits
- Les applications Java sont indépendantes du système d'exploitation instalé sur la machine (Windows, Linux, ...). Cette indépendance est assurée par l'utilisation d'une machine virtuelle qui a été développée pour la majorité des systèmes actuels.
- Enfin, l'utilisation des interfaces graphiques est géré par Java.

Les données de l'entrepôt sont stockées dans des SGBDs. Nous avons choisi d'utiliser le système Mysql Server, version 5.0.6. Il dispose du langage SQL et il est gratuit, performant et bien documenté.

| Lien | Indication |
|---|---|
| → 1 | - relie l'interface utilisateur à la base de documents XML<br>*signifie que l'utilisateur peut choisir une source ou une répertoire des documents XML.* |
| → 2 | - relie les documents XML après transformation et l'interface utilisateur<br>*signifie que l'utilisateur peut obtenir le répertoire des documents XML après transformation.* |
| → 3 | - relie l'interface utilisateur aux documents XML<br>*signifie que l'utilisateur peut sélectionner le nom du répertoire des documents XML afin d'extraire les nœuds feuilles de l'ensemble des documents dans le même répertoire.* |
| → 4 | - relie la table de stockage de données à l'interface utilisateur<br>*signifie que les tables de données sont accessibles à l'utilisateur* |
| → 5 | - relie l'interface utilisateur à la table de stockage de données<br>*signifie que l'utilisateur peut interroger les tables.* |
| → 6 | - relie le processus d'interrogation à l'interface utilisateur<br>*signifie que le processus d'interrogation fournit à l'utilisateur les résultats de sa requête.* |

TAB. 6.1 – Description détaillée de l'architecture fonctionnelle du système

Nous avons utilisé les packages et les librairies SWING pour développer l'interface graphique. Cette librairie permet de définir facilement des interfaces agréables à utiliser. Les packages de Java sont utilisés pour implémenter notre système de façon modulaire. Les principaux packages sont :

- le package Xalan de Apache qui permet de traiter les fichiers XML et d'utiliser des langages tels que XPath et XSLT.
- le package Mysql Connector, qui permet de se connecter à la base.

### 6.3.1 Module de Transformation

La figure 6.2 présente la partie de l'interface graphique de OSIX qui permet de transformer un répertoire de documents XML satisfaisant un schéma local en un répertoire de documents XML satisfaisant le schéma global. La fenêtre principale est composée de plusieurs boutons :

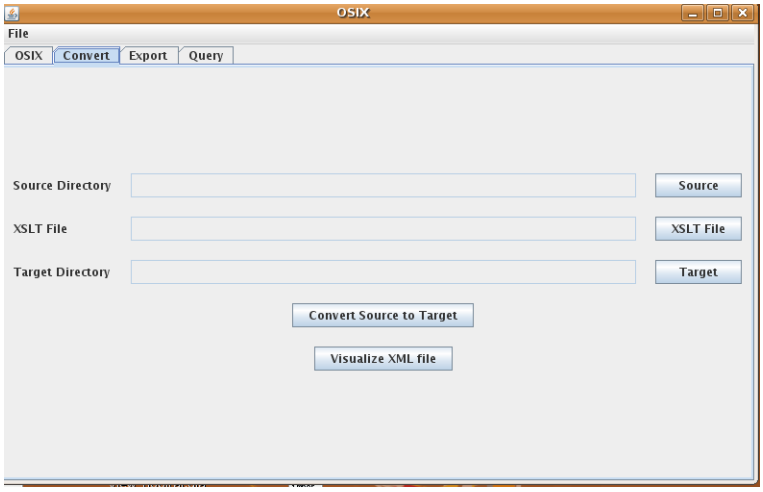


FIG. 6.2 – Fenêtre principale pour la tâche de transformation

Le bouton « **Source** » affiche une nouvelle fenêtre « *Select the source directory* », montrée dans la figure 6.3, qui permet de sélectionner le nom du répertoire contenant l'ensemble de documents XML satisfaisant un schéma local ; ce nom est visualisé dans le champ « *Source Directory* » de la fenêtre principale.

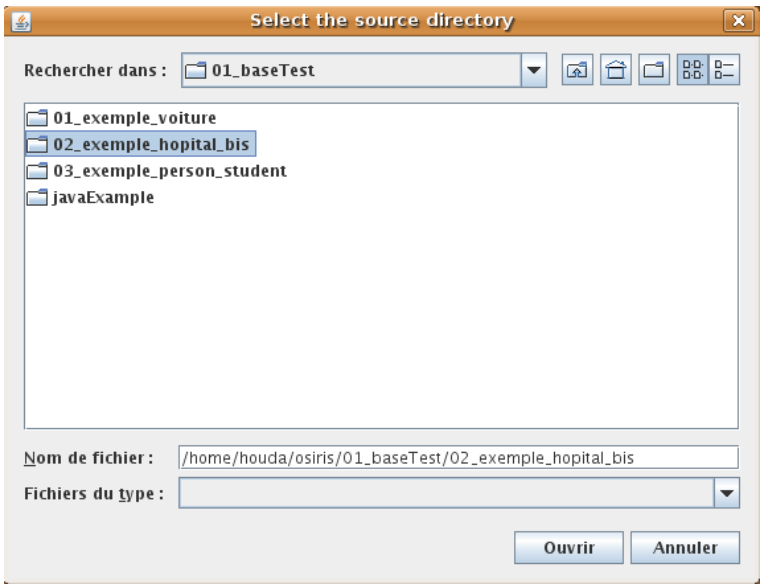


FIG. 6.3 – Fenêtre pour sélectionner le répertoire source

Le bouton « **XSLT FILE** » affiche une nouvelle fenêtre « *Select the XSLT File* » montrée dans la figure 6.4, qui permet de sélectionner le fichier XSLT qui fait le mapping entre le schéma local et le schéma global. Le nom du fichier XSLT est affiché dans le champ « *XSLT File* » da la fenêtre principale.

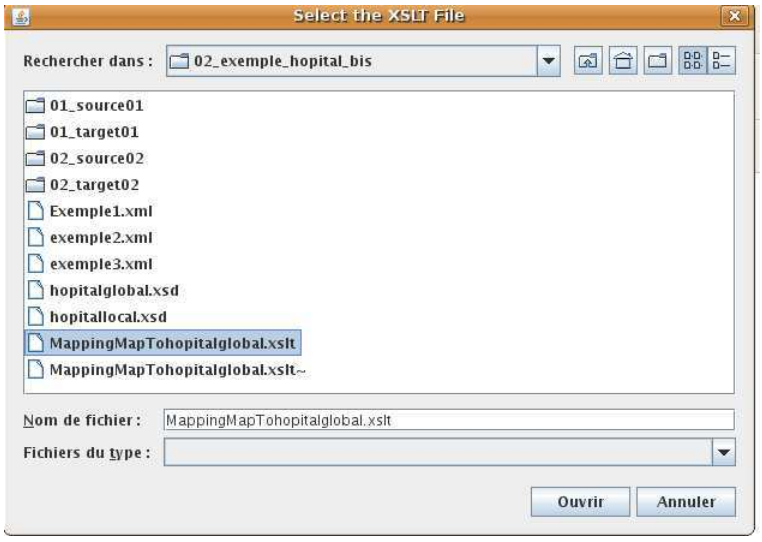


FIG. 6.4 – Fenêtre pour sélectionner le fichier XSLT

Le bouton « **Target** » affiche une nouvelle fenêtre « *Select the target directory* », montrée dans la figure 6.5. ELLe permet de spécifier le nom de répertoire où on veut stocker les documents

XML générés après la transformation.

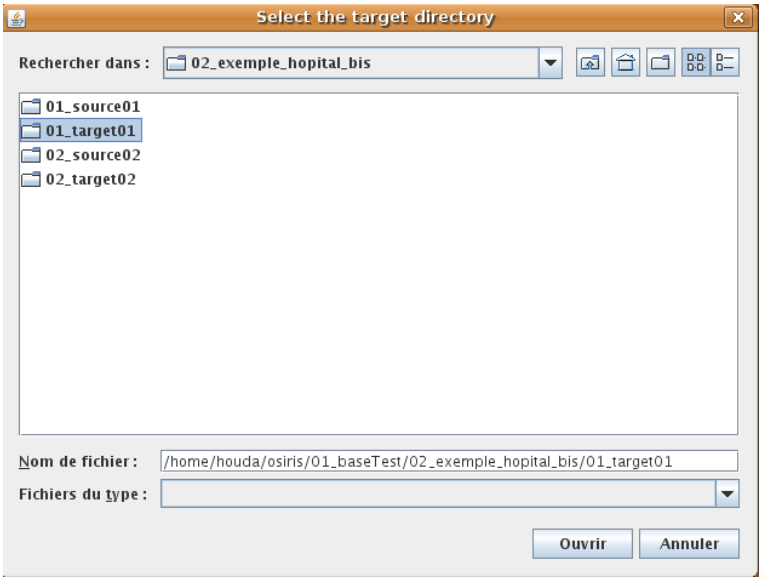


FIG. 6.5 – Fenêtre pour sélectionner le répertoire cible

Le bouton « **Visualize XML FILE** » permet d'afficher (visualiser) les fichiers XML après la transformation.

Le bouton « **Convert Source to Target** », qui est le bouton clé de cette interface. Il permet de faire la transformation des fichiers XML_L vers les fichiers XML_G en utilisant le mapping entre un schéma local et le schéma global ; ce mapping est exprimé par le fichier XSLT sélectionné. L'algorithme utilisé pour cette phase est présenté dans la section 6.4.1.

### 6.3.2 Module d'Extraction

La figure 6.6 présente la partie de l'interface graphiques de OSIX qui permet d'extraire les valeurs atomiques (nœuds feuilles) de fichiers XML après transformation ; elle permet aussi de choisir le nom pour la table relationnelle où on veut stocker les valeurs atomiques de Documents XML (les feuilles de l'arbre DOM) via le champ « *Table Name* ».

Cette partie de l'interface est composée des boutons suivants :

- Le bouton « **database connection** » permet de :
  1. Se connecter à la base de données en spécifiant le nom de l'utilisateur et le mot de passe via le bouton « **login** » (voir 6.7).
     - Un indicateur « **DB Status** » de l'état de connexion est grisé quand la connexion est établie.
  2. Se déconnecter de la base via le bouton « **logout** ».
- Le bouton « **Export to database** » permet de parcourir les fichiers XML, extraire les valeurs atomiques et stocker ces valeurs dans une table dynamique ayant le même nom que celui pris au début du traitement.

L'utilisation de cette interface se fait de la manière suivante :

1. Sélectionner « **Select Directory** ».
2. Saisir le nom de la table dans « **Table Name** ».

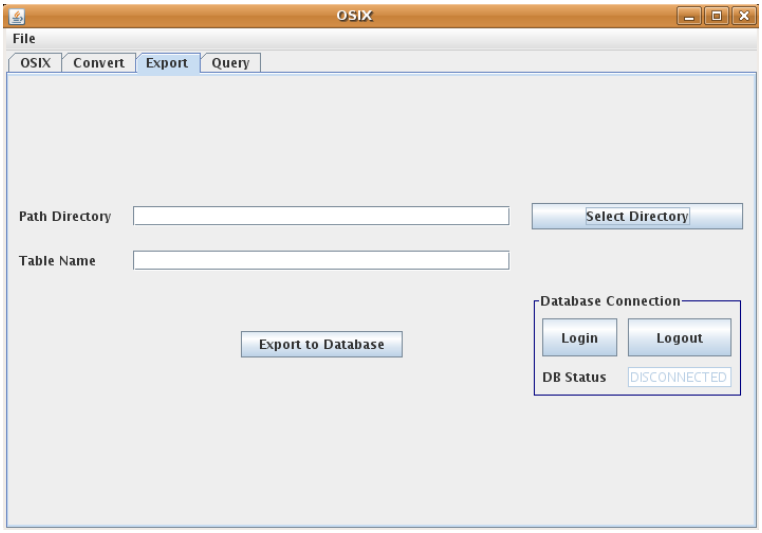


FIG. 6.6 – Fenêtre principale pour la tâche de l'extraction

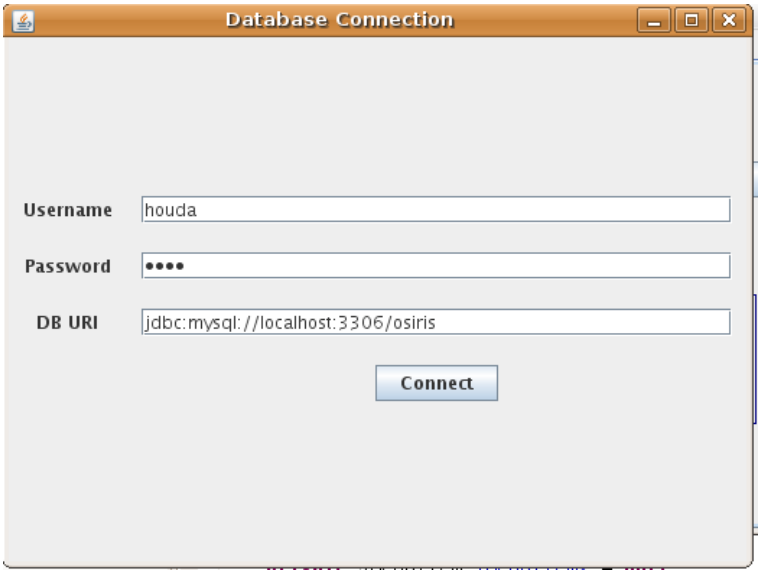


FIG. 6.7 – Fenêtre pour la connexion à la base de données

3. Cliquer sur « **Login** » pour se connecter.
4. Exécuter « **Export to Base Data** ». Pour afficher le contenu de la table, l'utilisateur peut interroger la base par une requête SQL.

Les algorithmes utilisés pour cette phase sont présentés dans la section 6.4.

### 6.3.3 Module d'Interrogation (*Query*)

La figure 6.8 présente la fenêtre principale de la partie Interrogation (*Query*) de l'outil OSIX. Cette partie permet de sélectionner ou saisir une requête SQL sur une source de documents XML et fournir la réponse appropriée.
La fenêtre principale est composée de :

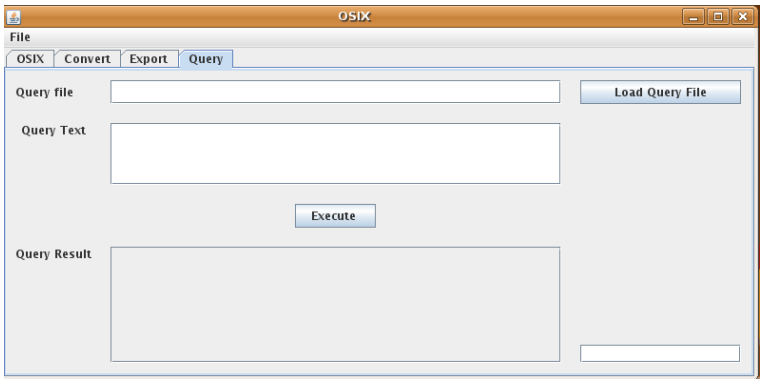


FIG. 6.8 – La fenêtre principale de l'interrogation

- Le bouton *"Load Query File"* permet d'ouvrir une nouvelle fenêtre intitulée *Select the Query* (voir 6.9). Cette fenêtre permet de sélectionner le nom de la requête qu'on veut exécuter. Le nom de la requête est affiché dans le champ *"Query file"* de la fenêtre principale. Le texte SQL de la requête sera affiché dans le champ "*Query text*" de la fenêtre principale.
- Le bouton "*Execute*" permet d'exécuter la requête sélectionnée et d'afficher le résultat dans le champ "*Query Result*". Par exemple, dans la figure 6.9, on a choisi la requête "PATIENT OU MEDECIN Masculin.sql". Le texte SQL de cette requête est affiché dans la figure 6.10.

Cette requête permet d'afficher le nom et l'adresse de tous les médecins et les patients masculins de l'hôpital. Le système va parcourir la table d'indexation fournie par le système Osiris afin d'extraire les oids d'objets qui sont valides pour la requête. C'est à dire les OIDs d'objets tels que (Patient = V ou Médecin= V) et m= V (V pour valide, m pour Masculin). Ensuite, le système va extraire les valeurs atomiques correspondantes dans la table Personne.

La fenêtre principale de la requête contient aussi un bouton "Chronomètre" qui permet d'afficher le temps d'exécution de la requête sélectionnée. Le temps est estimé en millisecondes.

L'utilisation de l'interface Interrogation (*Query*) doit se faire comme suit :

1. Sélectionner la requête via (*load Query File*) ou saisir la requête dans le champ *Query text*.
2. Exécuter la requête.

Un autre exemple de requête est montré dans la figure 6.11. L'utilisateur doit saisir une requête, par exemple afficher le nom et la spécialité des médecins agés de plus de 30 ans.

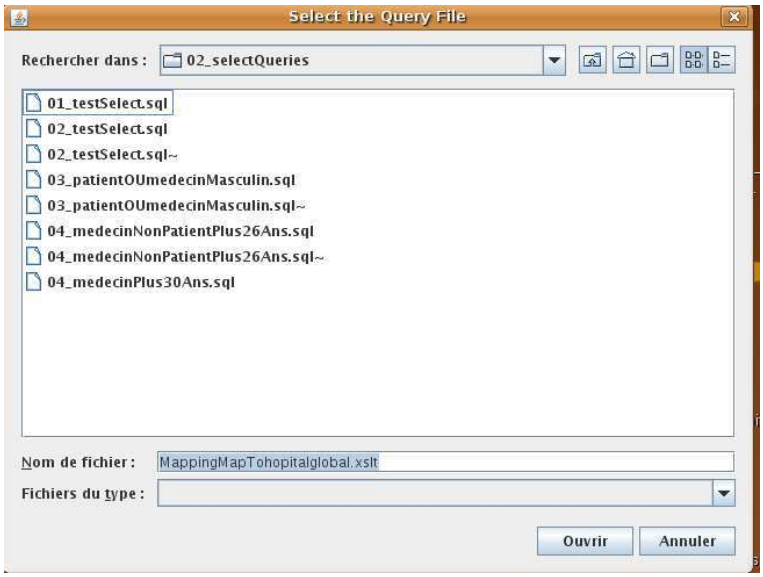


FIG. 6.9 – Fenêtre pour sélectionner une requête

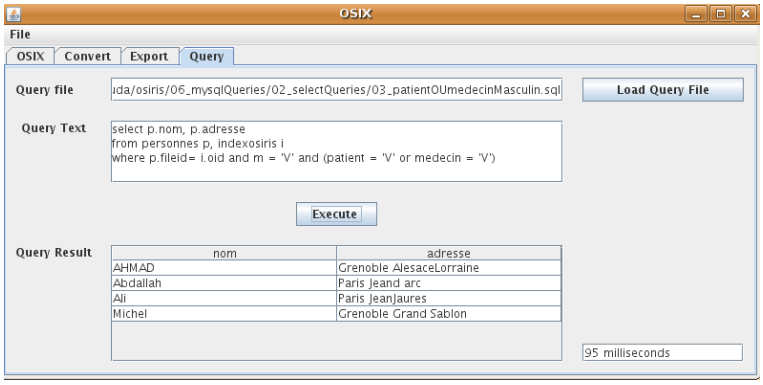


FIG. 6.10 – Résultat de la requête sélectionnée

Il interroge le système comme suit :

```
SELECT nom, spécialité
FROM Personne
WHERE age > 30
```

Le système va accèder à la table d'indexation fournie par le système Osiris pour récupérer les OIDs d'objets satisfaisant les vues MEDECIN et ADULTE, c'est-à-dire les OIDS d'objets telque MEDECIN=V et ADULTE=V. Ensuite, notre système va accéder à la table Personne pour récupérer les attributs demandés (nom et spécialité). Le résultat sera visualisé dans le champ " *Query Result*". L'algorithme utilisé pour cette phase est présenté dans la section 6.4.5.

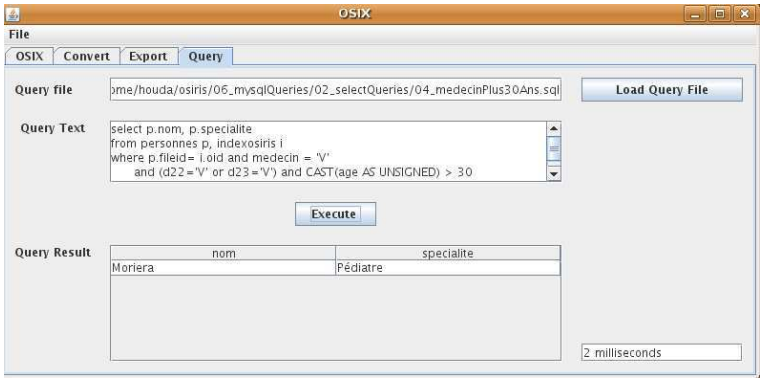


FIG. 6.11 – Résultat de la requête sélectionnée

## 6.4 Algorithmes

### 6.4.1 Algorithme de Transformation

Type DocFile : <doc : Document,cheminFichier : Chaîne, nomFichier : Chaîne>

TransformationDirectoryFiles(nomRepertoire :Châine ; nomFichierXSLT : Chaîne) : Liste(DocFile)

Algorithm 6.1 – algorithme de transformation

Debut
$resultat : Liste(DocFile)$ {variable résultat}
$fichierSource : Fichier$ ;
$ListeF : Liste(Fichier)$ ;
$ListeF \leftarrow ListeFichier(nomRepertoire)$ ;
**Pour** $chaquefichierSourcedansListeF$ **Faire**
  $doc : Document$ ;
  $docF : DocFile$ ;
  $doc \leftarrow parseXMLFile(fichierSource.cheminAbselut(), nomFichierXSLT)$ ;
  $docF \leftarrow crerDocFile(doc, fichierSource.chemin(), fichierSource.nom())$ ;
  $resultat.ajouter(docF)$ ;
**FinPour**
$retourner(resultat)$
Fin

TransformationDirectoryFiles (nR,nxslt) est une fonction du type Liste(Docfile), elle a deux paramètres : le nom du répertoire "nR" qui contient l'ensemble de fichiers source, le deuxième paramètre est un fichier "nxslt" qui définit la transformation.

Pour chaque fichier source, la transformation définie dans le fichier "nxslt" est appliquée sur tous les fichiers sources contenus dans le répertoire "nR". A la fin on obtient une liste de fichiers après transformation. Le noyau de cet algorithme est la fonction ParseXMLFile, qui fait la transformation effective d'un fichier (voir l'algorithme suivant 6.2).

ParseXMLFile(nomFichierSource :chaîne, nomFichierXSLT :chaîne) :Document

"parseXMLFile (nFSource,nFXSLT)" est une fonction qui retourne un document, elle a deux paramètres : un fichier source et la transformation définie dans le fichier "nFXSLT". Le résultat

Algorithm 6.2 – Transformation

Debut
$t : Transformeur$ ; {définit dans DOM}
$s : Source$ ; {définit dans DOM}
$doc : Document$ ;
$fichierSource : Fichier$ ;
$fichierSource \leftarrow créerFichier(nomFichierSource)$ ;
$doc \leftarrow créerSource(nomFichierXSLT)$ ;
$s \leftarrow t.transformer(fichierSource, s)$ ;
$retourner(doc)$ ;
Fin

est le document XML après transformation.

### 6.4.2 Algorithme d'Extraction

Type AttributeValue : <nomAttribut : chaîne, dfSource : DocFile, valeur : chaîne, Xpath :chaîne).

getAttributeValues (docF :DocFile) : liste(AttributeValue).

Algorithm 6.3 – Extraction

Debut
$resultat : Liste(AttributeVlue)$ ;
$doc : Document; Xpath : chaîne$ ;
$doc \leftarrow DdocF.getDocument()$ ;
Xpath ← "//*[not(*)]" ; {expression Xpath qui permet de récupèrer tous les noeuds feuilles de l'arbre doc}
$ListeNoeuds : Liste(Noeuds)$ ;
$ListeNoeuds \leftarrow XPathAPI.selectNodeListe(doc, Xpath)$ ;
**pour** *chaquenoeudn* **Faire**
  $nom : chaîne$ ;
  $valeur : chaîne$ ;
  $path : chaîne$ ;
  $nom \leftarrow n.getNom()$ ;
  $valeur \leftarrow getValue()$ ;
  $Path \leftarrow n.getXPath(n, doc)$ ;
  $Attr : AttributeValue$ ;
  $Attr \leftarrow créerAttributeValue(nom, valeur, path, docF)$ ;
  $result.ajouter(attr)$ ;
**FinPour**
*Retourner resultat* ;
Fin

Cet algorithme retourne une liste d'AttributsValue où chaque élément est un élément feuille de l'objet document contenu dans DocFile. Chaque feuille est définie par son nom, sa valeur, son chemin (Path) et le nom du document où se trouve. Il utilise l'algorithme suivant pour l'obtention du chemin de la feuille.

getXPath(n :Noeud ; doc :Document) :chaîne.

Algorithm 6.4 – Extraction

Debut
chemin :chaîne ;
$chemin \leftarrow$
**pour** chaque noeud $n1 \leftarrow n$ et n1$\neq$ doc **Faire**
  chaine Xpathlocal ;
  $Xpathlocal \leftarrow n.nom()$ ;
  $chemin \leftarrow Xpathlocal + " /" + chemin$ ;
**FinPour**
$chemin \leftarrow " /" + chemin$ ;
retourner cheminFichier
Fin

Cet algorithme retourne l'expression Xpath qui permet d'atteindre un nœud '**n**' dans un document '**doc**' (le chemin d'accès au nœud).

### 6.4.3 Algorithme pour l'accès à la base

ConnectionBD(nom : chaîne, motPasse : chaîne, urlBD : chaîne) : Connexion

Algorithm 6.5 – Accès à la base

**DEBUT**
*connect* : *Connexion* {variable résultat} ;
$d$ : $DriverManager$ ;
$dchangerDriver('databaseDriver')$ ; {variable intéermédiare}
$connect \leftarrow d.getConnection(nom, motPass, urlBD)$ ;
*retourner connect* ;
**FIN**

"ConnectBD (n, mp, url)" est une fonction qui permet à l'utilisateur de nom 'n' avec un mot de passe 'mp' de se connecter à la base de données localisée par l'URL 'url'.

### 6.4.4 Algorithme de stockage de données

remplirTable(listAttVal : Liste<AttributeValue>, connect : Connexion, nomTable : Chaîne)
remplirTable(lat, c, nt) est une fonction qui permet d'inserer les éléments d'une liste d'attributs 'lat' dans une table nommée 'nt' définie dans une base de données accessible par une connexion 'c'.

### 6.4.5 Algorithme d'Exécution des requêtes

ExecuterRequete (requete : Chaîne, connect : Connexion) : ResultSet ;
ExecuterRequete(req : chaîne, c : Connexion) est une fonction qui retourne un ensemble résultat de l'exécution de la requête 'req' sur une base de données accessible via la connexion 'c'.

## 6.5 Conclusion

Le système que nous avons implémenté a été présenté dans ce chapitre. Il permet d'interroger de données XML en se basant sur l'indexation du système Osiris afin de réduire au maximum le temps de réponse. Pour cela il utilise le mécanisme de vues défini par Osiris. Notre système a été développé en Java. Il contient dix-sept classes et 3927 lignes de code.

Pour l'interface graphique on a utilisé la bibliothèque SWING. Le SGBD relationnel utilisé pour le stockage des données est le système Mysql Server. L'exemple présenté dans le chapitre précédent (hôpital) a été complètement implémenté avec notre outil en utilisant un contrôle (chronomètre) pour calculer le temps d'exécution.

Algorithm 6.6 – Stockage de données

**DEBUT**
$requetInsertion : Chaîne$ ;
$partieValeur : Chaîne$ ;
$attV : AttributeValue$ ;
$atV : Chaîne$ ;
$atN : Chaîne$ ;
$statement : PrepareStatement$ ;
$requetInsertion \leftarrow'' INSERTINTO'' + nomTable +'' (fileID,''$ ;
partieValeur ← partieValeur + " ' " +listAttVal.get(0).getDfSource().getFilePath()+" ',"" ;
**TANT QUE** il existe d'autre attV **Faire**
  $atN \leftarrow attV.getAttributeName()$ ;
  $atV \leftarrow attV.getAttributeName()$ ;
  **SI** il existe d'autre attV **FAIRE**
    $requeteInsertion \leftarrow requeteInsertion + atN+''$,'' ;
    partieValeur ← partieValeur+ " ' "+atV+ ", " ;
  **SINON**
    $requeteInsertion \leftarrow requeteInsertion + atN+'')''$ ;
    partieValeur ← partieValeur+ " ' "+atV+ " ') " ;
  **ENDSI**
**FIN TANT QUE**
$requeteInsertion \leftarrow requeteInsertion + partieValeur$ ;
$statement \leftarrow connect.prepareStatement(requeteInsertion)$ ;
$statement.executeQuery(requeteInsertion)$ ;
$statement.execte(''commit'')$ ;
$statement.close()$ ;
**FIN**

Algorithm 6.7 – Execution de requêtes

DEBUT
$resultat : ResultSet$ ; {variable résultat}
$statement : PreparedStatement$ ;
$statement \leftarrow connect.PreparedStatement(requete)$ ;
$statement.executeQuery(req)$ ;
$resultatstatement.getResultatSet()$ ;
retourner résultat ;
FIN

# Conclusion générale

## 1 Bilan et contributions

Dans ce mémoire, nous avons présenté une approche matérialisée basée sur les vues pour l'intégration de documents XML en utilisant les principes d'Osiris, système de représentation de données et de connaissance de la famille des logiques de description. L'interêt d'utiliser le système Osiris dans notre approche est double. Tout d'abord, il permet de définir plusieurs points de vue sur une même famille d'objets ; ensuite, son système d'indexation permet l'optimisation sémantique des requêtes.

Une source XML peut contenir un grand nombre de documents XML avec des schémas plus au moins différents. Avec le modèle de données Osiris, il n'est pas nécessaire qu'un schéma XML corresponde à toutes les vues d'un P-type, mais seulement à un sous-ensemble de ses vues. Une conséquence imédiate est que des schémas très différents peuvent être en correspondance avec le même P-type. Par ailleurs, Osiris offre un système d'indexation multicritère puissant.

Dans cette thèse nous avons étudié l'apport des principales fonctionnalités d'Osiris - classement, indexation et optimisation sémantique des requêtes - à l'intégration de documents XML. Notre approche est basée principalement sur quatre processus : description, transformation, extraction et interrogation. Le processus de description produit un schéma XML cible (schéma abstrait), qui représente un schéma Osiris. Le processus de transformation réécrit chaque document XML, qui satisfait un schéma concret, en termes du schéma cible abstrait. Le processus d'extraction extrait les valeurs atomiques de chaque document obtenu après transformation. Les objets correspondant à ces valeurs sont alors classés et indexés. Le processus d'interrogation utilise le mécanisme d'optimisation sémantique des requêtes d'Osiris pour extraire les objets d'interêt pour une requête.

Notre approche utilise la méthode *GAV* pour définir le schéma d'intégration : on définit le schéma global d'Osiris comme un ensemble de vues sur les sources.On transforme ensuite chaque document source qui satisfait un schéma concret en un document satisfaisant le schéma abstrait. Le schéma global d'Osiris est défini par une hiérarchie de vues, où chaque vue est définie par ses vues mères, ses attributs et contraintes propres. Nous avons utilisé la norme XML schéma pour produire le schéma abstrait représentant le schéma Osiris, pour plusieurs raisons expliquées dans le chapitre 5 (section) ; parmi ces raisons on peut citer : XML schéma donne la possibilité de représenter les différents types de données, et permet de décrire les références d'un élément vers un autre élément.

Notre système utilise la structure d'indexation (ISDs) offerte par Osiris afin d'effectuer l'optimisation sémantique des requêtes. Cela permet au système de déterminer automatiquement à quelles vues appartient un document XML. En conséquence, l'espace de recherche pour les documents qui font partie de la réponse à la requête peut être réduit à l'espace des vues satisfaisant la requête. Cette optimisation est importante parce que l'exploitation actuelle d'un grand

nombre de documents reste un problème, même si le principe d'utilisation d'un schéma global abstrait constitue un progrès important pour l'exploitation des documents XML d'une manière homogène.

Pour le stockage des données, nous utilisons un SGBD relationnel (Mysql Server), ce qui permet de profiter des bonnes performances de ces systèmes. Nous stockons les informations suivantes : l'identificateur de l'objet dans le système Osiris, l'identificateur du document qui contient la donnée et les valeurs des attributs classificateurs d'Osiris et tous les autres attributs représentants l'ensemble de feuilles de l'arbre DOM de chaque document XML dans la source.

Notre méthode de stockage permet d'interroger l'entrepôt de données directement pour répondre aux requêtes de l'utilisateur sans avoir besoin de consulter les sources de données. Cela permet d'optimiser le temps de réponse de la requête.

Nous avons proposé un algorithme pour traiter le cas de la mise à jour d'une donnée d'un document XML dans la source. L'idée principale de cet algorithme est de stocker dans une mémoire temporaire (une table relationnelle) les méta-données pour la mise à jour qui contiennent la correspondance entre chaque donnée de la source et l'objet correspondant en Osiris. Quand la valeur d'une donnée d'un document dans la source change, la table "méta-données" fournit l'objet Osiris correspondant à cette donnée modifiée.

Enfin, pour valider notre approche, nous avons développé l'outil OSIXT (OSIX Tool), qui applique le système OSIX à l'intégration et l'interrogation de documents XML simulant les données d'un hôpital. Cet outil utilise les méthodes et les algorithmes que nous avons présentés dans notre mémoire. Il est écrit en langage Java et utilise le SGBD relationnel (Mysql Server) pour le stockage des données. Pour l'interface graphique, OSIXT utilise la bibliothèque SWING. L'outil OSIXT permet aussi de calculer le temps d'exécution d'une requête en utilisant un contrôle (chronomètre).

## 2 Perspectives

Cette thèse a ouvert plusieurs perspectives pour améliorer et compléter le travail angagé dans ce mémoire. Tout d'abord, il serait intéressant de comparer expérimentalement les performances de notre système avec les systèmes existants pour l'intégration et l'interrogation de documents XML. Ces mesures permettraient de quantifier l'apport en termes de performance de l'architecture et les algorithmes que nous avons proposés pour le stockage et l'interrogation de l'entrepôt de données. Il permet aussi de quantifier notre méthode pour l'optimisation sémantique de requêtes.

D'autres perspectives permettraient d'approfondir l'étude du problème de la mise à jour que nous avons présenté et d'étendre l'outil OSIXT proposé pour prendre en compte le cas de la mise à jour d'une donnée d'un document XML dans la source.

On peut aussi proposer d'étudier la capacité d'étendre le système OSIX pour passer à l'échelle, c'est à dire intégrer des sources de données à l'échelle du Web, parce que les sources de données XML disponibles sur le Web sont de plus en plus nombreuses.

# Annexe A

# Sources de données XML

***Document1 XML***

```
<?xml version=\textacutedbl1.0\textgravedbl encoding=\textacutedbl UTF-8\textgravedbl?>
<SAMU>M.H.1991
  <Individus>
    <CV>
      <nom-famille>Ali</nom-famille>
      <prenom>Omar</prenom>
      <ville>Paris</ville>
      <rue>Jean Jaures</rue>
      <sexe>M</sexe>
      <date_naissance>1981</date_naissance>
    </CV>
    <code_sector>S1</code_sector>
    <id>1P</id>
    <afection>Crdiaque</afection>
  </Individus>
 <Individus>
    <CV>
      <nom-famille>Abdallah</nom-famille>
      <prenom>Bassam</prenom>
      <viM.H.1991lle>Paris</ville>
      <rue>Jean d'Arc</rue>
      <sexe>M</sexe>
      <date_naissance>1943</date_naissance>
      <domaine_interet>Cardio</domaine_interet>
    </CV>
    <code_sector>S1</code_sector>
    <id>1M</id>
 </Individus>
 <Secteur>
   <num_sector>S1</num_sector>
   <nom_sector>Cardiologie</nom_sector>
   <responsable>1M</responsable>
 </Secteur>
```

```
 <Traitement>
  <code_tM.H.1991raitement>T1</code_traitement>
  <designation>Med1</desgnation>
  <place>Michalon</place>
  <value>260</value>
 </Traitement>
 <Soin>
   <codesoin>Soi1</codesoin>
   <codePat>1P</codePat>
   <codemed>1M</codemed>
   <numtraitement>T1</numtraitement>
   <date>2008</date>
 </Soin>
</SAMU>
```

***Document2 XML***

```
<?xml version=\textacutedbl1.0\textgravedbl encoding=\textacutedbl UTF-8\textgravedbl?>
<SAMU>
  <Individus>
    <CV>
      <nom-famille>Adam</nom-famille>
      <prenom>Janette</prenom>
      <ville>Grenoble</ville>
      <rue>Bon Pasteur</rue>
      <sexe>F</sexe>
      <date_naissance>1974</date_naissance>
    </CV>
    <code_sector>S3</code_sector>
    <id>2P</id>
    <afection>Alergie</afection>
  </Individus>
 <Individus>
    <CV>
      <nom-famille>Bernard</nom-famille>
      <prenom>Maryana</prenom>
      <ville>Grenoble</ville>
      <rue>Victor Hugo</rue>
      <sexe>F</sexe>
      <date_naissance>1978</date_naissance>
      <domaine_interet>radiologie</domaine_interet>
    </CV>
    <code_sector>S2</code_sector>
    <id>2M</id>
 </Individus>
 <Individus>
    <CV>
      <nom-famille>Moriera</nom-famille>
      <prenom>Janette</prenom>
```

```
      <ville>Grenoble</ville>
      <rue>Fleurs</rue>
      <sexe>F</sexe>
      <date_naissance>1958</date_naissance>
      <domaine_interet>Pediatre</domaine_interet>
    </CV>
    <code_sector>S3</code_sector>
    <id>3M</id>
 </Individus>
 <Secteur>
   <num_sector>S2</num_sector>
   <nom_sector>radiologie</nom_sector>
   <responsable>2M</responsable>
 </Secteur>
<Secteur>
   <num_sector>S3</num_sector>
   <nom_sector>Pediatre</nom_sector>
   <responsable>3M</responsable>
 </Secteur>
 <Traitement>
  <code_traitement>T2</code_traitement>
  <designation>Asbirin</desgnation>
  <place>Hopital Nord</place>
  <value>250</value>
 </Traitement>
<Traitement>
  <code_traitement>T3</code_traitement>
  <designation>Rocatain</desgnation>
  <place>Hopital Sud</place>
  <value>20</value>
 </Traitement>
 <Soin>
   <codesoin>Soi3</codesoin>
   <codePat>3P</codePat>
   <codemed>3M</codemed>
   <numtraitement>T3</numtraitement>
   <date>2007</date>
 </Soin>
</SAMU>
```

***Document3 XML***

```
<?xml version=\textacutedbl1.0\textgravedbl encoding=\textacutedbl UTF-8\textgravedbl?>
<SAMU>
  <Individus>
    <CV>
      <nom-famille>AHMAD</nom-famille>
      <prenom>Yehia</prenom>
```

```
      <ville>Grenoble</ville>
      <rue>Alesace Lorraine</rue>
      <sexe>M</sexe>
      <date_naissance>2007</date_naissance>
    </CV>
    <code_sector>S4</code_sector>
    <id>3P</id>
    <afection>Anpholoanza</afection>
  </Individus>
 <Individus>
    <CV>
      <nom-famille>Marchalle</nom-famille>
      <prenom>Selvie</prenom>
      <ville>Grenoble</ville>
      <rue>Peupliers</rue>
      <sexe>F</sexe>
      <date_naissance>2005</date_naissance>
      <domaine_interet>Cardio</domaine_interet>
    </CV>
    <code_sector>S4</code_sector>
    <id>4P</id>
    <afection>Gripe</afection>
 </Individus>
<Individus>
    <CV>
      <nom-famille>Michel</nom-famille>
      <prenom>Dimitry</prenom>
      <ville>Grenoble</ville>
      <rue>Grand Sblon/rue>
      <sexe>M</sexe>
      <date_naissance>1978</date_naissance>
      <domaine_interet>nephrologie</domaine_interet>
    </CV>
    <code_sector>S4</code_sector>
<id>4M</id>
 </Individus>
 <Secteur>
   <num_sector>S4</num_sector>
   <nom_sector>nephroloque</nom_sector>
   <responsable>4M</responsable>
 </Secteur>
</SAMU>
```

# Annexe B

# Schéma global d'un hôpital

```
<?xml version="1.0" encoding="UTF-8"?>
<xs:schema elementFormDefault="quaélified" attributeFormDefault="unqualified">
     <xs:element name="Hopital">
        <xs:annotation>
          <xs:documentation>Comment describing your root element</xs:documentation>
        </xs:annotation>
        <xs:complexType>
           <xs:sequence>
             <xs:element name="Personnes" maxOccurs="unbounded">
                <xs:complexType>
                   <xs:sequence>
                      <xs:element name="id" type="xs:string"/>
                      <xs:element name="nom" type="xs:string"/>
                      <xs:element name="prenom" type="xs:string"/>
                      <xs:element name="Adresse" type="xs:string" minOccurs="0"/>
                      <xs:element name="sexe" type="xs:string"/>
                      <xs:element name="age" type="xs:integer"/>
                      <xs:element name="specialite" type="xs:string" minOccurs="0"/>
                      <xs:element name="code_service" type="xs:string" minOccurs="0"/>
                      <xs:element name="Poste" type="xs:string" minOccurs="0"/>
                      <xs:element name="Salaire" type="xs:double" minOccurs="0"/>
                      <xs:element name="situation">
                         <xs:simpleType>
                            <xs:restriction base="xs:string">
                              <xs:enumeration value=""Medecin""/>
                               <xs:enumeration value=""Patient""/>
                               <xs:enumeration value=""Administrateur""/>
                               <xs:enumeration value=""/>
                            </xs:restriction>
                          </xs:simpleType>
                      </xs:element>
                      <xs:element name="maladie" type="xs:string" minOccurs="0"/>
                   </xs:sequence>
                </xs:complexType>
```

```
      </xs:element>
      <xs:element name="Service" maxOccurs="unbounded">
              <xs:complexType>
                   <xs:sequence>
                         <xs:element name="numserv" type="xs:string"/>
                         <xs:element name="nomserv" type="xs:string"/>
                         <xs:element name="Chef" type="xs:string"/>
                    </xs:sequence>
             </xs:complexType>
      </xs:element>
      <xs:element name="Soin" minOccurs="0">
             <xs:complexType>
                  <xs:sequence>
                       <xs:element name="numsoin" type="xs:string"/>
                       <xs:element name="numpat" type="xs:string"/>
                       <xs:element name="nummed" type="xs:string"/>
                       <xs:element name="refmed" type="xs:string"/>
                       <xs:element name="date_soin" type="xs:date"/>
                 </xs:sequence>
             </xs:complexType>
    </xs:element>
    <xs:element name="Sejours" minOccurs="0">
              <xs:complexType>
                 <xs:sequence>
                    <xs:element name="numsejour" type="xs:string"/>
                    <xs:element name="numpat" type="xs:string"/>
                    <xs:element name="numservice" type="xs:string"/>
                    <xs:element name="entree" type="xs:date"/>
                   <xs:element name="sortie" type="xs:date"/>
                 </xs:sequence>
              </xs:complexType>
       </xs:element>
 <xs:element name="medicaments" minOccurs="0">
        <xs:complexType>
            <xs:sequence>
                  <xs:element name="refmed" type="xs:string"/>
                  <xs:element name="designation" type="xs:string"/>
                  <xs:element name="laboratoire" type="xs:string"/>
                  <xs:element name="prix" type="xs:float"/>
           </xs:sequence>
         </xs:complexType>
</xs:element>
<xs:element name="realiser" minOccurs="0">
      <xs:complexType>
         <xs:sequence>
               <xs:element name="numrealise" type="xs:string"/>
               <xs:element name="numpat" type="xs:string"/>
               <xs:element name="numed" type="xs:string"/>
```

```
                    <xs:element name="numanl" type="xs:string"/>
                    <xs:element name="numres" type="xs:string"/>
                    <xs:element name="date_analyse"/>
                </xs:sequence>
            </xs:complexType>
      </xs:element>
      <xs:element name="resultats" minOccurs="0">
           <xs:complexType>
               <xs:sequence>
                      <xs:element name="numres" type="xs:string"/>
                      <xs:element name="libelle" type="xs:string"/>
              </xs:sequence>
          </xs:complexType>
    </xs:element>
    <xs:element name="analyses">
         <xs:complexType>
            <xs:sequence>
                    <xs:element name="numanl" type="xs:string"/>
                    <xs:element name="designation" type="xs:string"/>
                    <xs:element name="laboratoire" type="xs:integer"/>
                    <xs:element name="prix" type="xs:float"/>
            </xs:sequence>
        </xs:complexType>
   </xs:element>
  </xs:sequence>
</xs:complexType>
<xs:key name="id_service">
   <xs:selector xpath="Personnes"/>
   <xs:field xpath="code_service"/>
</xs:key>
<xs:keyref name="ref_service" refer="id_service">
   <xs:selector xpath="Service/numserv"/>
   <xs:field xpath="."/>
</xs:keyref>
<xs:key name="id_chef">
   <xs:selector xpath="Service"/>
   <xs:field xpath="Chef"/>
</xs:key>
<xs:keyref name="ref_medcin" refer="id_chef">
   <xs:selector xpath="Personnes/id"/>
   <xs:field xpath="."/>
</xs:keyref>
<xs:key name="id_patient">
    <xs:selector xpath="Soin"/>
    <xs:field xpath="numpat"/>
</xs:key>
<xs:keyref name="ref_patient" refer="id_patient">
    <xs:selector xpath="Personnes/id"/>
```

```
    <xs:field xpath="."/>
</xs:keyref>
<xs:key name="id-medcin">
   <xs:selector xpath="Soin"/>
   <xs:field xpath="nummed"/>
</xs:key>
<xs:keyref name="ref_medcin1" refer="id-medcin">
   <xs:selector xpath="Personnes/id"/>
   <xs:field xpath="."/>
</xs:keyref>
<xs:key name="id-medicament">
   <xs:selector xpath="Soin"/>
   <xs:field xpath="refmed"/>
</xs:key>
<xs:keyref name="ref_medicament" refer="id-medicament">
   <xs:selector xpath="medicaments/refmed"/>
   <xs:field xpath="."/>
</xs:keyref>
<xs:unique name="id-person">
   <xs:selector xpath="Personnes/id"/>
   <xs:field xpath="."/>
</xs:unique>
<xs:key name="idi-patient">
   <xs:selector xpath="Sejours"/>
   <xs:field xpath="numpat"/>
</xs:key>
<xs:keyref name="reff-patient" refer="idi-patient">
   <xs:selector xpath="Personnes/id"/>
   <xs:field xpath="."/>
</xs:keyref>

<xs:key name="idii-patient">
   <xs:selector xpath="realiser"/>
   <xs:field xpath="numpat"/>
</xs:key>
<xs:keyref name="refff-patient" refer="idii-patient">
   <xs:selector xpath="Personnes/id"/>
   <xs:field xpath="."/>
</xs:keyref>
<xs:key name="idi-medcin">
   <xs:selector xpath="realiser"/>
   <xs:field xpath="numed"/>
</xs:key>
<xs:keyref name="reff-medcin" refer="idi-medcin">
    <xs:selector xpath="Personnes/id"/>
    <xs:field xpath="."/>
</xs:keyref>
<xs:key name="id-analyse">
```

```
   <xs:selector xpath="realiser"/>
   <xs:field xpath="numanl"/>
</xs:key>
<xs:keyref name="ref-analyse" refer="id-analyse">
   <xs:selector xpath="analyses/numanl"/>
   <xs:field xpath="."/>
</xs:keyref>
<xs:key name="id-result">
   <xs:selector xpath="realiser"/>
   <xs:field xpath="numres"/>
</xs:key>
<xs:keyref name="ref-result" refer="id-result">
   <xs:selector xpath="resultats/numres"/>
   <xs:field xpath="."/>
</xs:keyref>
<xs:key name="id-servic">
  <xs:selector xpath="Sejours"/>
  <xs:field xpath="numservice"/>
</xs:key>
<xs:keyref name="service_ref" refer="id-servic">
   <xs:selector xpath="Service/numserv"/>
   <xs:field xpath="."/>
   </xs:keyref>
</xs:element>
</xs:schema>
```

# Annexe C

# Schéma local d'un hôpital

```
<?xml version="1.0" encoding="UTF-8"?>
<xs:schema elementFormDefault="qualified" attributeFormDefault="unqualified">
  <xs:element name="SAMU">
    <xs:annotation>
      <xs:documentation>Comment describing your root element</xs:documentation>
   </xs:annotation>
   <xs:complexType>
     <xs:sequence>
       <xs:element name="Individus" maxOccurs="unbounded">
          <xs:complexType>
           <xs:sequence>
            <xs:element name="Etat"/>
            <xs:element name="CV">
              <xs:complexType>
               <xs:sequence>
                <xs:element name="nom-famille" type="xs:string"/>
                <xs:element name="prenome" type="xs:string"/>
    o            <xs:element name="ville" type="xs:string"/>
                <xs:element name="rue" type="xs:string"/>
                <xs:element name="sexe" type="xs:string"/>
                <xs:element name="date_naissance" type="xs:integer"/>
                <xs:element name="domaine_interet" type="xs:string" minOccurs="0"/>
                <xs:element name="function" minOccurs="0"/>
              </xs:sequence>
             </xs:complexType>
       </xs:element>
       <xs:element name="code_sector" type="xs:string" minOccurs="0"/>
       <xs:element name="id" type="xs:string"/>
       <xs:element name="afection" minOccurs="0"/>
    </xs:sequence>
  </xs:complexType>
</xs:element>
<xs:element name="Secteur" minOccurs="0" maxOccurs="unbounded">
```

```
  <xs:complexType>
    <xs:sequence>
      <xs:element name="num_sector" type="xs:string"/>
      <xs:element name="nom_sector" type="xs:string"/>
      <xs:element name="responsable" type="xs:string"/>
   </xs:sequence>
  </xs:complexType>
</xs:element>
<xs:element name="Traitement" minOccurs="0" maxOccurs="unbounded">
  <xs:complexType>
    <xs:sequence>
     <xs:element name="code_traitement" type="xs:string"/>
     <xs:element name="designation" type="xs:string"/>
     <xs:element name="place" type="xs:string"/>
     <xs:element name="value" type="xs:double"/>
    </xs:sequence>
  </xs:complexType>
</xs:element>
<xs:element name="Soin" minOccurs="0" maxOccurs="unbounded">
   <xs:complexType>
     <xs:sequence>
      <xs:element name="codesoin" type="xs:string"/>
      <xs:element name="codePat" type="xs:string"/>
      <xs:element name="codemed" type="xs:string"/>
      <xs:element name="numtraitement" type="xs:string"/>
      <xs:element name="date" type="xs:string"/>
     </xs:sequence>
   </xs:complexType>
</xs:element>
</xs:sequence>
</xs:complexType>
<xs:key name="id_sector">
  <xs:selector xpath="Secteur"/>
  <xs:field xpath="num_sector"/>
</xs:key>
<xs:keyref name="ref_sector" refer="id_sector">
  <xs:selector xpath="Individus/code_sector"/>
  <xs:field xpath="."/>
</xs:keyref>
<xs:key name="id_person">
  <xs:selector xpath="Individus"/>
  <xs:field xpath="id"/>
</xs:key>
<xs:keyref name="ref_person" refer="id_person">
  <xs:selector xpath="Secteur/responsable"/>
  <xs:field xpath="."/>
</xs:keyref>
<xs:key name="id_pat">
```

```
  <xs:selector xpath="Soin"/>
  <xs:field xpath="codePat"/>
</xs:key>
<xs:keyref name="ref_pat" refer="id_pat">
  <xs:selector xpath="Individus/id"/>
  <xs:field xpath="."/>
</xs:keyref>
<xs:key name="id_med">
   <xs:selector xpath="Soin"/>
   <xs:field xpath="codemed"/>
</xs:key>
<xs:keyref name="ref_med" refer="id_med">
  <xs:selector xpath="Individus/id"/>
  <xs:field xpath="."/>
</xs:keyref>
<xs:key name="id_trait">
  <xs:selector xpath="Soin"/>
  <xs:field xpath="numtraitement"/>
</xs:key>
<xs:keyref name="ref_trait" refer="id_trait">
  <xs:selector xpath="Traitement/code_traitement"/>
  <xs:field xpath="."/>
</xs:keyref>
</xs:element>
</xs:schema>
```

# Annexe D

# Fichier XSLT contenant le mapping entre le schéma local et le schéma global d'un hôpital

```
<?xml version="1.0" encoding="UTF-8"?>
<xsl:stylesheet version="2.0" exclude-result-prefixes="fn grp xs xsi xsl">
<xsl:output method="xml" encoding="UTF-8" indent="yes"/>
<xsl:template match="/">
    <Hopital>
      <xsl:attribute name="xsi:noNamespaceSchemaLocation" separator=" ">
          <xsl:sequence select="'C:/DOCUME~1/ahmad/Bureau/Exemple_Hopital/hopitalglobal.xsd
      </xsl:attribute>
      <xsl:variable name="var1_instance" as="node()" select="."/>
      <xsl:for-each select="$var1_instance/SAMU">
        <xsl:for-each select="Individus">
          <Personnes>
            <id>
             <xsl:sequence select="xs:string(id)"/>
           </id>
           <nom>
             <xsl:sequence select="xs:string(CV/nom-famille)"/>
          </nom>
          <prenom>
            <xsl:sequence select="xs:string(CV/prenome)"/>
         </prenom>
         <Adresse>
           <xsl:sequence select="fn:concat(fn:concat(xs:string(CV/ville), ' '), xs:string(C
        </Adresse>
        <sexe>
          <xsl:sequence select="xs:string(CV/sexe)"/>
       </sexe>
       <age>
        <xsl:sequence select="xs:string(xs:integer((xs:decimal('2008') - xs:decimal(xs:inte
      </age>
```

```
      <xsl:for-each select="CV/domaine_interet">
        <specialite>
          <xsl:sequence select="xs:string(.)"/>
        </specialite>
    </xsl:for-each>
    <xsl:for-each select="code_sector">
       <code_service>
          <xsl:sequence select="xs:string(.)"/>
       </code_service>
   </xsl:for-each>
   <xsl:for-each select="CV/function">
      <Poste>
        <xsl:sequence select="xs:string(.)"/>
     </Poste>
  </xsl:for-each>
  <situation>
    <xsl:sequence select="xs:string(Etat)"/>
</situation>
<xsl:for-each select="afection">
   <maladie>
      <xsl:sequence select="xs:string(.)"/>
   </maladie>
</xsl:for-each>
</Personnes>
</xsl:for-each>
<xsl:for-each select="Secteur">
  <Service>
    <numserv>
         <xsl:sequence select="xs:string(num_sector)"/>
   </numserv>
   <nomserv>
        <xsl:sequence select="xs:string(nom_sector)"/>
  </nomserv>
  <Chef>
        <xsl:sequence select="xs:string(responsable)"/>
  </Chef>
</Service>
</xsl:for-each>
<xsl:for-each select="Soin">
   <Soin>
    <numsoin>
      <xsl:sequence select="xs:string(codesoin)"/>
    </numsoin>
    <numpat>
      <xsl:sequence select="xs:string(codePat)"/>
   </numpat>
   <nummed>
      <xsl:sequence select="xs:string(codemed)"/>
```

```
   </nummed>
   <refmed>
    <xsl:sequence select="xs:string(numtraitement)"/>
  </refmed>
  <date_soin>
    <xsl:sequence select="xs:string(xs:date(xs:string(date)))"/>
  </date_soin>
</Soin>
</xsl:for-each>
<xsl:for-each select="Traitement">
   <medicaments>
     <refmed>
       <xsl:sequence select="xs:string(code_traitement)"/>
    </refmed>
    <designation>
      <xsl:sequence select="xs:string(designation)"/>
    </designation>
    <laboratoire>
      <xsl:sequence select="xs:string(place)"/>
    </laboratoire>
    <prix>
      <xsl:sequence select="xs:string(xs:float(xs:double(value)))"/>
   </prix>
  </medicaments>
</xsl:for-each>
</xsl:for-each>
</Hopital>
</xsl:template>
</xsl:stylesheet>
```

# Bibliographie


[Abiteboul et al., 2000] Abiteboul, S., Buneman, P., & Suciu, D. (2000). *Data on the Web : From Relations to Semistructured Data and XML.* San Francisco, California : Morgan Kaufmann.

[Abiteboul et al., 1995] Abiteboul, S., Hull, R., & Vianu, V. (1995). *Foundations of databases.* Addison-Wesley.

[Abiteboul et al., 1996] Abiteboul, S., Quass, D., McHugh, J., Widom, J., & Wiener, J. L., Eds. (1996). *The Lorel Query Language for Semistructured Data*, volume 1. International Journal on Digital Libraries. (pp. 68-88).

[Abiteboul et al., 1998] Abiteboul, S., Widom, J., & Lahiri, T. (1998). A unified approach for querying structured data and xml. http ://www.w3.org/TandS/QL/QL98/pp/serge.html.

[Adali et al., 1996] Adali, S., C, K. S., Papakonstantinou, Y., & Subrahmanian, V. S. (1996). Query caching and optimization in distributed mediator systems. *SIGMOD Rec.*, 25(2), 137–146.

[Agarwal et al., 1996] Agarwal, S., Agrawal, R., Deshpande, P., Gupta, A., Naughton, J. F., Ramakrishnan, R., & Sarawagi, S. (1996). On the computation of multidimensional aggregates. In T. M. Vijayaraman, A. P. Buchmann, C. Mohan, & N. L. Sarda (Eds.), *VLDB'96, Proceedings of 22th International Conference on Very Large Data Bases, September 3-6, 1996, Mumbai (Bombay), India* (pp. 506–521). : Morgan Kaufmann.

[Aguilera, 2001] Aguilera, V. (2001). *Interrogation de Documents XML.* PhD thesis, Université Joseph Fourier.

[Aguilera et al., 2002] Aguilera, V., Cluet, S., Milo, T., Veltri, P., & Vodislav, D. (2002). Views in a large-scale xml repository. *The VLDB Journal*, 11(3), 238–255.

[Arens et al., 1994] Arens, Y., Chee, C. Y., Hsu, C.-N., , In, H., & Knoblock, C. A., Eds. (1994). *Query Processing in an Information Mediator.*

[Arens et al., 1998] Arens, Y., Hsu, C. N., & Knoblock, C. A. (1998). Query processing in the sims information mediator. (pp. 82–90).

[Baril, 2003] Baril, X. (2003). *Un modèle de vues pour l'intégration de sources de données XML : VIMIX.* PhD thesis, l'Université des Sciences et Techniques du Languedoc.

[Baril, 2006] Baril, X. (2006). *Incremental Method for XML View Maintenance in Case of Non Monitored Data Sources*, chapter Lecture Notes in Computer Science, (pp. 148–157). Springer-Verlag Berlin Heidelberg.

[Baru et al., 1999] Baru, C., Gupta, A., Ludäscher, B., Marciano, R., Papakonstantinou, Y., Velikhov, P., & Chu, V. (1999). Xml-based information mediation with mix. *SIGMOD Rec.*, 28(2), 597–599.

[BASSOLET, 1992] BASSOLET, C. G. (1992). *Approches connexionnistes du classement en Osiris. Vers un classement probabiliste.* PhD thesis, Joseph Fourier - Grenoble1.

[Batini et al., 1986] Batini, C., Lenzerini, M., & Navathe, M. (1986). A comparative analaysis of methodologies for database schema integration. *ACM Computing Surveys*, 18, 323–364.

[Bertino & Ferrari, 2001] Bertino, E. & Ferrari, E. (2001). Xml and data integration. *IEEE Internet Computing*, 5(6), 75–76.

[Bertino & Guerrini, 1995] Bertino, E. & Guerrini, G. (1995). Objects with multiple most specific classes. In *ECOOP '95 : Proceeding of the 9th European Conference on Object-Oriented Programming* London, UK : Springer-Verlag.

[Billel, 2005] Billel, G. (2005). Optimisation de requêtes xquery à travers des vues. http ://perso.telecom-paristech.fr/ gueni/pdf/MsterRepport-2005.pdf.

[Bosak et al., 1998] Bosak, J., Bray, T., Connolly, D., Maller, E., Nicol, G., Sperberg-MacQueen, C., Wood, L., & Clark, J. (1998). W3c xml specification dtd.

[Boyd et al., 2004] Boyd, M., Kittivoravitkul, S., & Lazanitis, C. (2004). Automed : A bav data integration system for heterogeneous data sources. In *The 16th International Conference Advanced Information System Engineering.*

[Bray et al., 1998] Bray, T., Paoli, J., & Sperberg-MacQueen, C. (1998). Extensible markup language (xml). 1.0 (W3C Recommendation).

[Breitbart et al., 1986] Breitbart, Y., Olson, P. L., & Thompson, G. R. (1986). Database integration in a distributed heterogeneous database system. In *Proceedings of the Second International Conference on Data Engineering* (pp. 301–310). Washington, DC, USA : IEEE Computer Society.

[Brien & Poulovassilis, 2003] Brien, P. & Poulovassilis, A. (2003). Data integration by bi-directional schema transformation rules. In *ICDE.*

[Calvanese et al., 2000] Calvanese, D., Giacomo, G. D., Lenzerini, M., & Vardi, M. Y. (2000). What is view-based query rewriting ? In *Knowledge Representation Meets Databases* (pp. 17–27).

[Cannataro et al., 2005] Cannataro, M., Cluet, S., Tradigo, G., Veltri, P., & Vodislav, D. (2005). Using views to query xml. In *Encyclopedia of Database Technologies and Applications* (pp. 729–735).

[Cattell, 1993] Cattell, R. (1993). *Object Databases.* Morgan Kaufmann.

[Cattell et al., 1997] Cattell, R.-G., Barry, D.-K., Bartels, D., Berler, M., Eastman, J., Gamerman, S., Jordan, D., Springer, A., Strickland, H., & Wade, D. (1997). *The object database standard : ODMG 2.0.* San Francisco, CA, USA : Morgan Kaufmann Publishers Inc.

[Chamberlin et al., 2001] Chamberlin, D., Robie, J., & Florescu, D. (2001). Quilt : An xml query language for heterogeneous data sources. In *Selected papers from the Third International Workshop WebDB 2000 on The World Wide Web and Databases* (pp. 1–25). London, UK : Springer-Verlag.

[Chawathe et al., 1994] Chawathe, S., Garcia-Molina, H., Hammer, J., Ireland, K., Papakonstantinou, Y., Ullman, J., & Widom, J. (1994). The TSIMMIS project : Integration of heterogeneous information sources. In *16th Meeting of the Information Processing Society of Japan* (pp. 7–18). Tokyo, Japan.

[Christophides et al., 1994] Christophides, V., S. Abiteboul, S. C., & Scholl, M. (1994). From structured documents to novel query facilities. In *ACM SIGMOD Conf. on Management of Data* (pp. 313–324). Minneapolis, Minnesota.

[Cluet et al., 1998] Cluet, S., Delobel, C., Siméon, J., & Smaga, K. (1998). Your mediators need data conversion ! In *SIGMOD '98 : Proceedings of the 1998 ACM SIGMOD international conference on Management of data* (pp. 177–188). New York, NY, USA : ACM.

[Cluet et al., 2001] Cluet, S., Veltri, P., & Vodislav, D. (2001). Views in a large scale xml repository. In *VLDB '01 : Proceedings of the 27th International Conference on Very Large Data Bases* (pp. 271–280). San Francisco, CA, USA : Morgan Kaufmann Publishers Inc.

[Cooper et al., 2001] Cooper, B., SAMPLE, N., & Franklin, M. (2001). A fast index for semistructured data. In *VLDB '01 : Proceedings of the 27th International Conference on Very Large Data Bases* (pp. 341–350). San Francisco, CA, USA : Morgan Kaufmann Publishers Inc.

[Coulet, 2008] Coulet, A. (2008). *Construction et utilisation d'une base de connaissances pharmacogénomique pour l'intégration de données et la découvetre de connaissance.* PhD thesis, ?Université Henri Poincaré.

[Czejdo et al., 1987] Czejdo, B., Rusinkiewicz, M., & Embley, D. (1987). An approach to schema integration and query formulation in federated database systems. In *Proceedings of the Third International Conference on Data Engineering* (pp. 477–484). Washington, DC, USA : IEEE Computer Society.

[Deutsch et al., 1998] Deutsch, A., Fernandez, M., Florescu, D., Levy, A., Maier, D., & Suciu, D. (1998). Xml-ql. In QL'98 [QL'98]. Electronic edition at http ://www.w3.org/TandS/QL/QL98/.

[Deutsch et al., 1999] Deutsch, A., Fernandez, M., Florescu, D., Levy, A. L., & Suciu, D. (1999). Xml-ql : A query language for xml. http ://www.w3.org/TR/NOTE-xml-ql/.

[Deutsch & Tannen, 2003] Deutsch, A. & Tannen, V. (2003). Reformulation of xml queries and constraints. In *9th International Conference on Database Theory.*

[Diallo, 2006] Diallo, G. (2006). *Une Architecture à Base d'Ontologies pour la Gestion Unifiée des Données Structurées et non Structurées.* PhD thesis, l'Université Joseph Fourier.

[Dittrich & Jonscher, 2000] Dittrich, K. R. & Jonscher, D. (2000). All together now : Towards integrating the world's information systems. In *JISBD* (pp.7̃).

[Fernandez & Suciu, 1998] Fernandez, M. F. & Suciu, D. (1998). Optimizing regular path expressions using graph schemas. In *ICDE* (pp. 14–23).

[Florescu & Kossmann, 1999] Florescu, D. & Kossmann, D. (1999). Storing and quering xml data using an rdbms. http ://www.soe.ucsc.edu/classes/cmps290s/Spring03/fk.pdf.

[Fundulaki et al., 2002] Fundulaki, I., Amann, B., Beeri, C., Scholle, M., & Vercoustre, A. M. (2002). Styx : Connecting the xml web to the world of semantics. In *EDBT* (pp. 759–761).

[Gardarin, 2002] Gardarin, G. (2002). *XML Des bases de données aux services Web.* Dunod.

[Gardarin & Yoon, 1996a] Gardarin, G. & Yoon, S. (1996a). Hyoql : Aquery language for structured hypermedia documents. In *The International Journal of Microcomputer Applications.*

[Gardarin & Yoon, 1996b] Gardarin, G. & Yoon, S. (1996b). Hyoql : Aquery language for structured hypermedia documents. *The onternational Journal od Microcomputer applications.*

[Gay, 1999] Gay, E. (1999). Vues osi sur une base relationnelle. Master's thesis, Mémoire d'ingénieur CNAM, Centre de Grenoble.

[Goh et al., 1999] Goh, C., Bressan, S., Madnick, S., & Siegel, M. (1999). Context interchange : new features and formalisms for the intelligent integration of information. *ACM Transactions on Information Systems*, 17(3), 270–270.

[Goldfarb, 1990] Goldfarb, C. F. (1990). *The SGML handbook*. New York, NY, USA : Oxford University Press, Inc.

[Goldman & Widom, 1997] Goldman, R. & Widom, J. (1997). Dataguides : Enabling query formulation and optimization in semistructured databases. In M. Jarke, M. J. Carey, K. R. Dittrich, F. H. Lochovsky, P. Loucopoulos, & M. A. Jeusfeld (Eds.), *VLDB'97, Proceedings of 23rd International Conference on Very Large Data Bases, August 25-29, 1997, Athens, Greece* (pp. 436–445). : Morgan Kaufmann.

[Hacid & Reynaud, 2004] Hacid, M. S. & Reynaud, C. (2004). L'integration de sources de donnees . *Revue Information - Interaction - Intelligence (R I3)*, 4(2).

[Halevy, 1999] Halevy, A. (1999). More on data management for xml. white paper, available online at : http ://www.cs.washington.edu/homes/alon/widom-response.html.

[Halevy, 2001] Halevy, A. (2001). Answering queries using views : A survey. *The VLDB Journal*, 10(4), 270–294.

[Hammer et al., 1995] Hammer, J., Garcia-Molina, H., Ireland, K., Papakonstantinou, Y., Ullman, J., & Widom, J. (1995). Information translation, mediation, and mosaic-based browsing in the tsimmis system. *Exhibits Program of the proceeding of the ACM SIGMOD International Conference on Management of data*, 24(2), 483.

[Hopcroft et al., 2001] Hopcroft, J. E., Motwani, R., & Ullman, J. D. (2001). *Introduction to automata theory, languages, and computation, 2nd edition*, volume 32. New York, NY, USA : ACM Press.

[Ives et al., 2002] Ives, Z. G., Halevy, A. Y., & Weld, D. S. (2002). An xml query engine for network-bound data. *The VLDB Journal*, 11(4), 380–402.

[Kashyap & Sheth, 1996] Kashyap, V. & Sheth, A. P. (1996). Semantic and schematic similarities between database objects : A context-based approach. *VLDB Journal : Very Large Data Bases*, 5(4), 276–304.

[Kermanshahani, 2008] Kermanshahani, S. (2008). Semi-materialized framework : a hybrid approach to data integration. In *CSTST '08 : Proceedings of the 5th international conference on Soft computing as transdisciplinary science and technology* (pp. 600–606). New York, NY, USA : ACM.

[Kim & Seo, 1991] Kim, W. & Seo, J. (1991). Classifying schematic and data heterogeneity in multidatabase systems. *Computer*, 24(12), 12–18.

[Kirk et al., 1995] Kirk, T., Levy, A. Y., Sagiv, Y., & Srivastava, D. (1995). The Information Manifold. In C. Knoblock & A. Levy (Eds.), *Information Gathering from Heterogeneous, Distributed Environments* Stanford University, Stanford, California.

[Krishnamurthy et al., 2003] Krishnamurthy, R., Kaushik, R., & Naughton, J. (2003). *Database and XML Technologies*, chapter XML-to-SQL Query Translation Literature : The State of the Art and Open Problems, (pp. 1–18). Springer Berlin / Heidelberg.

[Laurent et al., 2003] Laurent, M., Denilson, B., & Pierangelo, V. (2003). The xml web : a first study. In *WWW '03 : Proceedings of the 12th international conference on World Wide Web* (pp. 500–510). New York, NY, USA : ACM Press.

[Lenzerini, 2002] Lenzerini, M. (2002). Data integration : A theoretical perspective. In *Proceeding in the International Conference on Database Theory(PODS)* : ACM Press.

[Levy et al., 1996] Levy, A., Rajamaran, A., & J.Ordille (1996). Querying heterogeneous information sources using source description. In *VLDB'96* (pp. 252–262).

[Madhavan et al., 2001] Madhavan, J., Bernstein, P. A., & Rahm, E. (2001). Generic schema matching with cupid. In *VLDB '01 : Proceedings of the 27th International Conference on Very Large Data Bases* (pp. 49–58). San Francisco, CA, USA : Morgan Kaufmann Publishers Inc.

[Maitre et al., 1997] Maitre, J. L., Murisasco, E., & Rolbert, M. (1997). From annotated corpora to databaese : The sgmlql language. In J. Nerbonne (Ed.), *Linguistic Databases* (pp. 37–58). Standford, Califonia.

[Manolescu et al., 2001] Manolescu, L., Florescu, D., & Kossmann, D. (2001). Answering XML queries over heterogeneous data sources. In *VLDB'01* (pp. 241–250).

[McBrien & Poulovassilis, 2003] McBrien, P. & Poulovassilis, A. (2003). Defining peer-to-peer data integration using both as views rules. In *In proc.DBISP2P*.

[McHugh et al., 1997] McHugh, J., Abiteboul, S., Goldman, R., Quass, D., & Widom, J. (1997). Lore : A database management system for semistructured data. *SIGMOD Record*, 26(3), 54–66.

[Melton & Buxton, 2006] Melton, J. & Buxton, S. (2006). *Querying XML*. Morgan Kaufmann.

[Mena et al., 2000] Mena, E., Illarramendi, A., Kashyap, V., & Sheth, A. P. (2000). Observer : An approach for query processing in global information systems based on interoperation across pre-existing ontologies. *Distributed and Parallel Databases*, 8(2), 223–271.

[Milo & Suciu, 1999] Milo, T. & Suciu, D. (1999). Index structures for path expressions. In *ICDT '99 : Proceedings of the 7th International Conference on Database Theory* (pp. 277–295). London, UK : Springer-Verlag.

[M.Roth & Schwarz, 1997] M.Roth & Schwarz, P. (1997). Don't scrap it, wrap it ! a wrapper architecture for legacy data sources. In *VLDB '97 : Proceedings of the 23rd International Conference on Very Large Data Bases* (pp. 266–275). San Francisco, CA, USA : Morgan Kaufmann Publishers Inc.

[Nestorov et al., 1997] Nestorov, S., Ullman, J. D., Wiener, J. L., & Chawathe, S. (1997). Representative objects : Concise representations of semistructured, hierarchial data. In *ICDE '97 : Proceedings of the Thirteenth International Conference on Data Engineering* (pp. 79–90). Washington, DC, USA : IEEE Computer Society.

[NGOS, 2003] NGOS, T. D. (2003). *Fédération des données semi-structurées avec XML*. PhD thesis, Université de Versailles Saint Quentin-en-Yvelines.

[Papakonstantinou et al., 1995] Papakonstantinou, Y., Garcia-Molina, H., & Widom, J. (1995). Object exchange across heterogeneous information sources. In P. S. Yu & A. L. P. Chen (Eds.), *11th Conference on Data Engineering* (pp. 251–260). Taipei, Taiwan : IEEE Computer Society.

[Papakonstantinou & Velikhov, 1999] Papakonstantinou, Y. & Velikhov, P. (1999). Enhancing semistructured data mediators with document type definitions. In *Conference on Data Engineering (ICDE)* (pp. 136–145).

[Papakonstantinou & Vianu, 2000] Papakonstantinou, Y. & Vianu, V. (2000). DTD Inference for Views of XML Data. In *Proceedings of the Nineteenth ACM SIGMOD-SIGACT-SIGART Symposium on Principles of Database Systems* (pp. 35–46). Dallas, Texas.

[Pottinger & Halevy, 2001] Pottinger, R. & Halevy, A. (2001). Minicon : A scalable algorithm for answering queries using views. *The VLDB Journal*, 10(2-3), 182–198.

[Rahm & Bernstein, 2001] Rahm, E. & Bernstein, P. A. (2001). A survey of approaches to automatic schema matching. *The VLDB Journal*, 10(4), 334–350.

[Reynaud et al., 2001] Reynaud, C., Sirot, J. P., & Vodislav, D. (2001). Semantic integration of xml heterogeneous data sources. In *IDEAS '01 : Proceedings of the 2001 International*

*Symposium on Database Engineering & Applications* (pp. 199). Washington, DC, USA : IEEE Computer Society.

[Rousset & Reynaud, 2003] Rousset, M. & Reynaud, C. (2003). *Picsel and Xyleme : two illustrative information integration agents* . Book chapter in Intelligent Information Agents Research and Development in Europe, Springer-Verlag.

[Rousset et al., 2002] Rousset, M.-C., Bidault, A., Froidevaux, C., Galiardi, H., Goasdoue, F., Reynaud, C., & Safar, B. (2002). Construction de médiateurs pour intégrer des sources d'information multiples et hétérogènes : le projet picsel. *I3*, vol.2. n°1, 5–95.

[Sattler et al., 2005] Sattler, K.-U., Geist, I., & E.Schallehn (2005). Concept-based quering in mediator systems. *VLDB*.

[Scholl et al., 1991] Scholl, M., Laasch, C., & Tresch, M. (1991). Updatable views in object-oriented databases. In *Proc.2nd DOOD conf* (pp. 187–198).

[Sebi, 2007] Sebi, I. (2007). *Interrogation de Documents XML à Travers des Vues.* PhD thesis, EDITE, Laboratoire CEDRIC.

[Sheth & Kashyap, 1992] Sheth, A. P. & Kashyap, V. (1992). So far (schematically) yet so near (semantically). In *DS-5* (pp. 283–312).

[Sheth & Larson, 1990] Sheth, A. P. & Larson, J. A. (1990). Federated database systems for managing distributed, heterogeneous, and autonomous databases. *ACM Comput. Surv.*, 22(3), 183–236.

[Simonet, 1984] Simonet, A. (1984). *Types Abstraits et Bases de Données : formalisation du concept de partage et analyse statique de contraintes d'intégrité.* PhD thesis, Université Scientifique et Médicale de Grenoble, France.

[Simonet, 1996] Simonet, A. (1996). Classement d'objets et evaluation de requêtes en osiris. In *Bases de Données Avancées* (pp. 273–288). Cassis, France.

[Siméon, 1999] Siméon, J. (1999). *Intégration de sources de données hétérogènes.* PhD thesis, Université Paris XI.

[Solar & Doucet, 2002] Solar, G. V. & Doucet, A., Eds. (2002). *Médiation de données : solutions et problèmes ouverts*, Nancy. Actes des 2ème assises nationales du GDR I3.

[Stanat & McAllister, 1977] Stanat, D. & McAllister, D. (1977). *Discrete Mathematics in Computer Science.* Prentice Hall Professional Technical Reference.

[Suciu, 2000] Suciu, D. (2000). Semistructured data and xml. *Information organization and databases : foundations of data organization*, (pp. 9–30).

[Templeton et al., 1995] Templeton, M., H.Henley, Maros, E., & Buer, D. V. (1995). Interviso :dealing with the complexity of federated database access. *VLDB*, 4(2), 287–317.

[Thompson et al., 2001] Thompson, H., Beech, D., Maloney, M., & Mendelsohn, N. (2001). Xml schema part 1 : Structure.

[TOPER, 2007] TOPER, N. (2007). Interrogation de documents xml. Master's thesis, CNAM, Paris.

[UCSD & SDSC, 1999] UCSD & SDSC (1999). Mix - mediation of information using xml. Web site of the project. http ://xml.coverpages.org/mix.html.

[Ullman, 1988] Ullman, J. D. (1988). *Principles of database and knowledge-base systems.* Computer Science Press.

[Ullman, 2000] Ullman, J. D. (2000). Information integration using logical views. *Theoretical Computer Science*, 239(2), 189–210.

[Vodislav et al., 2005] Vodislav, D., Cluet, S., Corona, G., & Sebi, I. (2005). Xyview : Universal relations revisited. *BDA*.

[WANG et al., 2004] WANG, T., YANG, D., S.TANG, & LIU, Y. (2004). Discovering and generating materialized xml views in data integration system. *IDEAS '04 : Prceedings of the International Database Engineering and Application Symposium*, (pp. 395–400).

[Widerhold, 1992] Widerhold, G. (1992). Mediators in the architecture of future information system. *IEEE Computer Magazine*, 25(3), 38–49.

[Widom, 1999] Widom, J. (1999). Data management for xml : Research directions. *IEEE Data Eng.*, 22(3), 44–52. available online at http ://infolab.stanford.edu/ widom/xml-whitepaper.html.

[Wu & Buchmann, 1997] Wu, M.-C. & Buchmann, A. P. (1997). Research issues in data warehousing. In *Datenbanksysteme in Buro, Technik und Wissenschaft* (pp. 61–82).

[Xu & Embley, 2004] Xu, L. & Embley, D. W. (2004). Combining the best of global-as-view for data integration. In *ISTA*.

[Xyleme, 2001] Xyleme, S. A. (2001). Xyleme system. Retrieved from http ://www.xyleme.com.

[Yoon et al., 2001] Yoon, J.-P., Raghavan, V., & Chakilam, V. (2001). Bitcube : A three-dimensional bitmap indexing for xml documents. In *SSDBM '01 : Proceedings of the 13th International Conference on Scientific and Statistical Database Management* (pp. 158). Washington, DC, USA : IEEE Computer Society.

[Ziegler & Dittrich, 2004] Ziegler, P. & Dittrich, K. R., Eds. (2004). *Three decades of data integration - all problems solved ?*, volume 12, Toulouse, France. 18th IFIP World Computer Congress (WCC 2004).

# Résumé

Les données semi-structurées occupent une place croissante dans l'évolution du Web par le biais du langage XML. La gestion de telles données ne s'appuie pas sur un schéma prédéfini, comme dans le cas de données structurées, gérées par exemple par le modèle relationnel. Le schéma de chaque document est auto-contenu dans le document même, et des documents similaires peuvent être représentés par des schémas différents. C'est pourquoi les algorithmes et les techniques d'intégration et d'interrogation de telles sources de données sont souvent plus complexes que ceux définis pour l'intégration et l'interrogation de sources de données structurées.

L'objectif de notre travail est l'intégration de données XML en utilisant les principes d'Osiris, un prototype de SGBD-BC, dont le concept central est celui de vue. Dans ce système, une famille d'objets est définie par une hiérarchie de vues, où chaque vue est définie par ses vues mères, ses attributs et contraintes propres. Osiris appartient à la famille des logiques de description, la vue minimale d'une famille d'objets étant assimilée à un concept primitif et ses autres vues à des concepts définis. Un objet d'une famille satisfait certaines de ses vues. Pour chaque famille d'objets, Osiris construit, par analyse des contraintes définies dans toutes ses vues, un espace de classement n-dimensionnel. Cet espace sert de support au classement d'objets et aussi à leur indexation.

Dans cette thèse nous avons étudié l'apport des principales fonctionnalités d'Osiris - classement, indexation et optimisation sémantique des requêtes à l'intégration de documents XML. Pour cela nous produisons un schéma cible (XML schema abstrait), qui représente un schéma Osiris ; chaque document satisfaisant un schéma source (XML schema concret) est réécrit en termes de le schéma cible avant de subir l'extraction des valeurs de ses entités. Les objets correspondant à ces entités sont alors classés et indexés. Le mécanisme d'optimisation sémantique des requêtes d'Osiris peut dès lors être utilisé pour extraire les objets d'intérêt pour une requête.

Nous avons réalisé un prototype, nommé OSIX (Osiris-based System for the Integration of XML sources) et nous l'avons appliqué à l'intégration et l'interrogation de documents XML simulant les données d'un hôpital.


**Mots-clés:** Intégration de données, XML, modèle de vues, entrepôt de données.

# Abstract

Semi-structured data play an increasing role in the development of the Web through the use of XML. However, the management of semi-structured data poses specific problems because semi-structured data, contrary to classical databases, do not rely on a predefined schema. The schema of a document is contained in the document itself and similar documents may be represented by different schemas. Consequently, the techniques and algorithms used for querying or integrating this data are more complex than those used for structured data.

The objective of our work is the integration of XML data by using the principles of Osiris, a prototype of KB-DBMS, in which views are a central concept. In this system, a family of objects is defined by a hierarchy of views, where a view is defined by its parent views and its own attributes and constraints. Osiris belongs to the family of Description Logics ; the minimal view of a family of objects is assimilated to a primitive concept and its other views to defined concepts. An object of a family satisfies some of its views. For each family of objects, Osiris builds a n-dimensional classification space by analysing the constraints defined in all of its views. This space is used for object classification and indexation.

In this thesis we study the contribution of the main features of Osiris - classification, indexation and semantic query optimization - to the integration of XML documents. For this purpose we produce a target schema (an abstract XML schema), which represents an Osiris schema ; every document satisfying a source schema (concrete XML schema) is rewritten in terms of the target schema before undergoing the extraction of the values of its entities. The objects corresponding to these entities are then classified and indexed. The Osiris mechanism for semantic query optimization can then be used to extract the objects of interest of a query.

We have realized a prototype, named OSIX (Osiris-based System for the Integration of XML documents) and we have applied it to the integration and interrogation of XML documents simulating the data of a hospital.



Keywords : Data integration, XML, semi-structured data, views, data warehouse, semantic query optimization.