الجمهورية الجزائرية الديمقراطية الشعبية
République Algérienne Démocratique et Populaire
وزارة التعليم العالي والبحث العلمي
Ministère de l'enseignement supérieur et de la recherche scientifique

Université 20 Août 1955 Skikda
Faculté des Sciences
Département d'Informatique
Réf : D02114003M

جامعة 20 أوت 1955 سكيكدة
كلية العـلوم
قسم : الإعلام الآلـــي

MÉMOIRE

En vue de l'obtention du diplôme de

MAGISTER EN INFORMATIQUE

Option : ***Informatique Parallèle et Distribuée***

Spécialité : ***Techniques Avancées pour les Systèmes Parallèles et Distribués***

Intégration des Sources de Données Hétérogènes à Base Ontologique

Application à l'intégration de BDD.s Universitaires

Présenté par :

Samir SELLAMI

Soutenu publiquement : 08 Février 2015

Devant la Commission du Jury composé de :

JURY

Président :	REDJIMI Mohammed - Maître de Conférences A - Université 20 Août 55 - Skikda
Rapporteur :	BENCHIKHA-MAGRA Fouzia - Professeur - Université Abdelhamid Mehri - Constantine
Examinateurs :	ZAROUR Nacer eddine - Professeur - Université Abdelhamid Mehri - Constantine
	MAZOUZI Smaine - Maître de Conférences A - Université 20 Août 55 - Skikda

REMERCIEMENTS

Je tiens à adresser mes sincères remerciements à Madame Fouzia MAGRA-BENCHIKHA, pour m'avoir proposé ce sujet et avoir dirigé mes travaux, pour les conseils qu'elle n'a cessé de me prodiguer et surtout pour ses qualités humaines et scientifiques, son encadrement privilégié, sa patience et ses encouragements.

Que le Docteur M.REDJIMI, trouve ici l'expression de ma profonde reconnaissance et de ma très haute considération pour l'honneur qu'il m'a fait en acceptant de présider le jury.

Tous mes remerciements accompagnés de ma gratitude vont aux membres du jury Messieurs : N.ZAROUR et S.MAZOUZI, pour la célérité avec laquelle ils ont examiné, corrigé et critiqué ce travail.

A Monsieur KISSOUM Yacine pour ces consiles et son aide durant ma formation de magistère.
A Monsieur AMICH Ahcen et au Melle Soumya Kasri pour leurs efforts.

À tous ceux, qui ont contribué de près ou de loin pour leurs informations et conseils judicieux, de m'avoir accordé leurs précieux temps, à l'élaboration du présent mémoire.

DEDICACES

En témoignage de ma reconnaissance, de mon estime et de mon respect. En ce solo du jour qui clôt le cycle de mes études qu'il me soit permis de dédier ce mémoire :

A la source d'affection et d'amour, ma mère, pour sa bienveillance et ses sacrifices.
A la source de courage, mon père, dont le soutien moral était ma seule source d'énergie.

A la mémoire de mes grands père et mère Kaci et Fatima Zohra.
A la mémoire de mes grands père et mère Houssine et Baya.
A la mémoire de mon chère ami Hamza KAOUEN.

A mes sœurs et mes frères,
L'expression de ma reconnaissance de m'avoir accordé une grande attention, amour, encouragements, conseils et aide afin d'accomplir mes études.

Aux Membres de ma Promotion Magistère informatique 2011/2014 :
Sofiane, Riad, Meriem, Fatema et Souhila.

A mes ami(e)s Nabil, Karim, Driss, Didine, Amine, Lotfi, Nadir, Hako, Kalile, Rida, Fathi, Salim, Omar, Zino, Radouen, Halim, Mohamed, Latifa, Nesrine, Ikram, pour leur gentillesse.

À l'inestimable chère amie Fattoum à qui je dois une très grande gratitude.
A tous ceux qui m'ont aidé de près ou de loin.

A VOUS.

Samir ...

La connaissance est un arbre de vie qui grandit à la lumière des autres.

Le Pavernel.

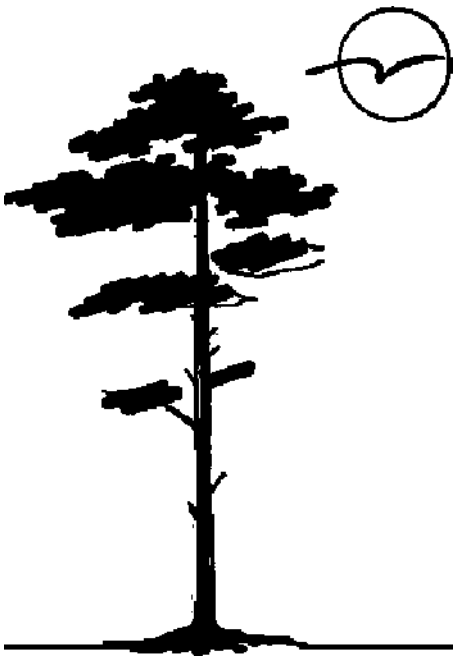


Table des matières

Remerciements ..... **ii**
Dédicaces ..... **iii**
Table des matières ..... **v**
Liste des Figures ..... **ix**
Résumé ..... **xi**
Introduction générale **1**
1. Problématique 2
2. Solution et Approche 3
3. Organisation du mémoire 5
Chapitre I Intégration des sources de données **8**
1. Introduction 9
2. Problématique de l'intégration de données 10
2.1 Autonomie des sources 10
2.2 Interdépendance des sources 10
2.3 Hétérogénéités des sources 11
2.3.1 Conflits de représentation 11
2.3.2 Conflits de noms (termes) 12
2.3.3 Conflits de contextes 12
2.3.4 Conflits de mesure de valeur 12
3. Systèmes d'intégration de données 13
3.1 Définition et Composante 13
3.2 Processus d'intégration 14
3.3 Tâche d'un système d'intégration 15
4. Classification des systèmes d'intégration 17
4.1 Architecture d'intégration / Localisation de données intégrées 17
4.1.1 Systèmes Multi-bases 17
4.1.2 Systèmes Fédérés 18
4.1.3 Architecture Matérialisée (Entrepôts de Données) 18
4.1.4 Architecture Virtuelle (Systèmes Médiateurs) 20
4.1.5 Architecture Mixte (Médiateurs / Entrepôt de données) 22
4.1.6 Systèmes P2P (Pair à Pair) 23
4.1.7 Discussion 23
4.2 Sens de mise en correspondance entre schéma global et schémas locaux 24
4.2.1 Global-as-View (GaV) 24

4.2.2 Local-as-View (LaV) 24
4.2.3 Generalized Local-as-View (GLaV) 25
4.2.4 BYU Global Local-as-View (BGLaV) 26
4.2.5 Both-as-View (BaV) 26
4.2.6 Discussion 26
4.3 Nature du processus d'intégration 27
4.3.1 Intégration manuelle de données 27
4.3.2 Intégration semi-automatique à l'aide d'ontologies linguistiques (ou thésaurus) 27
4.3.3 Intégration automatique à l'aide d'ontologie conceptuelle 29
4.3.4 Discussion 31
4.4 Langages de représentation de données intégrées 31
4.4.1 Systèmes fondés sur un schéma global à base de règles 31
4.4.2 Systèmes fondés sur un schéma global à base de classes 31
4.4.3 Systèmes fondés sur un schéma global à base d'arbres 31
4.5 Niveaux d'abstraction 32
4.6 Comparaison entre les différentes approches d'intégration 32
5. Traitement des requêtes dans un système d'intégration 33
6. Principaux systèmes d'intégration 34
6.1 Systèmes d'intégration 34
6.2 Comparaison des principaux systèmes d'intégration 35
7. Positionnement de notre approche 37
8. Conclusion 38
Chapitre II Technologie des ontologies et intégration des données 40
1. Introduction 41
2. La notion d'ontologie 42
2.1 Définition 42
2.2 Ontologies en Informatique 42
2.2.1 Définition 42
2.2.2 Concepts primitifs et concepts définis 43
3. Exemples de formalismes d'ontologies 44
3.1 Formalismes d'ontologies orientés gestion et échange de données 44
3.1.1 RDF/ RDF-Schéma 45
3.1.2 PLIB 49
3.2 Formalismes d'ontologies orientés inférence 52
3.2.1 DAML-ONT, OIL, DAML+OIL 53
3.2.2 OWL 53
4. Similitudes et différences des formalismes d'ontologie 57
4.1 Similitudes 57
4.2 Différences 57

4.3 Le modèle en oignon 58
4.3.1 Ontologies Conceptuelles Canoniques 59
4.3.2 Ontologies Conceptuelles Non-canoniques 60
4.3.3 Ontologies Linguistiques 61
5. Ontologie conceptuelle et Intégration des données 62
5.1 Différence entre ontologie et modèle conceptuel 62
5.2 Utilisation d'ontologies conceptuelles pour l'intégration de données 64
5.2.1 Intégration sémantique *à posteriori* 65
5.2.2 Intégration sémantique *à priori* 68
6. Conclusion 71
Chapitre III Approche d'intégration à base ontologique proposée 73
1. Introduction 74
2. Approche d'intégration de données 75
2.1 Approche de notre système de médiation 75
2.2 Ontologie de domaine 79
2.3 Modèle de représentation uniforme de l'ontologie de domaine 79
2.4 Description des sources de données à intégrer 82
3. Architecture du système 83
3.1 Module de création des requêtes 84
3.1.1 Sélection de l'ontologie de domaine 84
3.1.2 Création de requête 84
3.2 Serveur d'ontologie et métadonnées des sources 84
3.2.1 Informations sur l'ontologie de domaine 84
3.2.2 Informations sur les sources de données (métadonnées) 85
3.2.3 Choix de source de données locale 85
3.3 Module de mapping sémantique 85
3.4 Module de traitement des requêtes 86
3.4.1 La reformulation de la requête de l'utilisateur 86
3.4.2 Création du plan de requête 86
3.5 Module d'extraction des données 87
3.5.1 Traduction de requête de l'utilisateur au langage de requête de l'adaptateur 87
3.5.2 Extraction des données 87
3.6 Module de corrélation des données 87
3.6.1 Conversion des valeurs de données 87
3.6.2 Corrélation des réponses 87
3.6.3 Présentation du résultat final 87
4. Création de requête utilisateur 88
5. Les différents cas de mapping 89
6. Schéma de mapping 91

6.1 Exemple de mapping 92
7. Discussion 95
8. Conclusion 96
Chapitre IV Implémentation et mise en œuvre 98
1. Introduction 99
2. Création de l'ontologie de domaine de l'université 100
2.1 Méthodologie de construction des ontologies de domaine 100
2.2 Développement de l'ontologie de domaine de l'université 102
3. Interface utilisateur graphique 107
3.1 Barre Barre de Navigation à gauche 108
3.2 Le Header des Traitements en haut 108
3.3 Détails en milieu 108
4. Module Ontology Integration Perspective 108
4.1 Implémentation des Translators 112
4.2 Implémentation des Adaptateurs 113
4.3 Implémentation de l'Exportateur 113
5. Module des Sources de données 115
6. Module information de Mapping 117
6.1 Implémentation de module Mapping 117
6.2 Implémentation des Filtres 119
6.3 Implémentation des Convertisseurs 119
7. Module Reporting 119
8. Conclusion 119
Conclusion générale et perspectives 121
1. Conclusion 122
2. Contributions 122
3. Perspectives 124
Bibliographie 126
Glossaire 133
Annexe A : Codage en OWL de l'Ontologie Université. 135
Annexe B : Scripts DDL de création des sources de données. 142
Annexe C : Code ASP.Net C# d'implémentation de l'interface HERO-Med. 147

Liste des Figures

Fig. 1.1 – Système d'intégration de données. 13
Fig. 1.2 – Processus d'intégration. 14
Fig. 1.3 – Intégration par l'entrepôt. 18
Fig. 1.4 – Architecture du système WHIPS pour la maintenance de l'entrepôt de données. 20
Fig. 1.5 – Intégration par le médiateur. 20
Fig. 1.6 – Exemple de données OEM. 22
Fig. 1.7 – Architecture du système InfoMaster. 24
Fig. 1.8 – Exemple du sens de mise en correspondance entre schéma global et schéma local [33]. 25
Fig. 1.9 – Un exemple de thésaurus [58]. 29
Fig. 1.10 – Différentes architectures d'intégration à base ontologique proposées par Wache et al. [70]. 30
Fig. 1.11 – Positionnement de notre approche. 37
Fig. 2.1 – Exemple d'un graphe RDF. 46
Fig. 2.2 – Exemple d'ontologie exprimée en RDF schéma. 48
Fig. 2.3 – Le modèle en oignon. 58
Fig. 2.4 – Utilisation d'ontologies conceptuelles dans l'approche d'intégration sémantique a posteriori. 65
Fig. 2.5 – L'architecture de OBSERVER. 66
Fig. 2.6 – Architecture du système d'intégration proposée par le projet KRAFT. 67
Fig. 2.7 – Utilisation d'ontologies conceptuelles dans l'approche d'intégration sémantique a priori. 68
Fig. 2.8 – Architecture du système médiateur PICSEL. 69
Fig. 3.1 – Approche de notre système de médiation. 76
Fig. 3.2 – Une partie hiérarchique des concepts de l'ontologie de domaine (Exemple d'une ontologie de domaine). 77
Fig. 3.3 – Une partie des tables de source de donnée 1 (Exemple de source de donnée). 77
Fig. 3.4 – Architecture d'ontologie proposée pour résoudre les conflits sémantiques. 78
Fig. 3.5 – Modèle relationnel interne de représentation de l'ontologie de domaine. 81
Fig. 3.6 – Modèle relationnel logique des métadonnées des sources de données locales. 82
Fig. 3.7 – Architecture de système. 83
Fig. 3.8 – Schéma de Mapping. 92
Fig. 3.9 – Exemple de source de donnée local. 93
Fig. 3.10 – Une partie de l'ontologie de domaine Université. 93
Fig. 4.1 – Étapes de création des ontologies de domaine. 101
Fig. 4.2 – Hiérarchie des concepts de l'ontologie de domaine de l'Université. 104

Fig. 4.3 – Editeur d'ontologie « Protégé ». 106
Fig. 4.4 – Le modèle de représentation uniforme de l'ontologie de l'Université. 106
Fig. 4.5 – L'interface utilisateur graphique « GUI » du Prototype HERO-Med. 107
Fig. 4.6 – L'interface d'affichage de détail du concept « Student » de l'ontologie. 108
Fig. 4.7 – Exemple de création de requête sur le concept «Professor» de l'ontologie de domaine. 110
Fig. 4.8 – GUI et exemple de sauvegarde de la requête utilisateur. 111
Fig. 4.9 – Interface de transtation de requête globale en deux sous-requêtes. 112
Fig. 4.10 – Reformulation de requête et exportation des resultas final. 113
Fig. 4.11 – Interface de mis à jour du modèle de l'ontologie de domaine. 114
Fig. 4.12 – Interface de visualisation du modèle de graphe de l'ontologie de domaine 114
Fig. 4.13 – Exemple de sources de données hétérogènes 115
Fig. 4.14 – Interface des métadonnées des sources 116
Fig. 4.15 – Interface de mapping du Prototype HERO-Med 117
Fig. 4.16 – Tableaux de bord (dashboards) du Prototype HERO-Med 118

Liste des tableaux

Tab. 1.1 – Taxonomie de conflits -Parent et Spaccapietra 96-[8]. 12
Tab. 1.2 – Comparaison entre les différentes approches d'intégration. 32
Tab. 1.3 – Comparaison des principaux systèmes d'intégration. 36
Tab. 2.1 – Couche canonique des formalismes d'ontologies PLIB, RDFS et OWL. 60
Tab. 2.2 – Couche non-canonique des formalismes d'ontologies PLIB, RDFS et OWL. 61
Tab. 2.3 – Couche linguistique des formalismes d'ontologies PLIB, RDFS et OWL. 61
Tab. 3.1 – Classification des cas de mapping pour les concepts. 90
Tab. 3.2 – Classification des cas de mapping pour les Attributs. 90
Tab. 3.3 – La Table *concept_map* de d'exemple. 94
Tab. 3.4 – La Table *Views* de d'exemple. 94
Tab. 3.5 – La Table *attribut_map* de d'exemple. 94

Résumé

Avec le développement d'Internet et des intranets, l'échange et le partage de l'information provenant de diverses sources de données reparties, autonomes et hétérogènes deviennent un besoin crucial. Dans un tel contexte, il est souvent nécessaire pour une application d'accéder simultanément à plusieurs sources, du fait qu'elles contiennent des informations pertinentes et complémentaires. Pour ce faire, la solution des systèmes d'intégration a été proposée. Elle consiste à fournir une interface uniforme et transparente aux données pertinentes via une vue globale. Cependant, il existe de nombreux types d'hétérogénéité et différences entre les sources de données qui rendent l'intégration des données un processus difficile. L'hétérogénéité sémantique présente un défi majeur, chacun peut avoir un point de vue différent sur le même concept. Afin de réduire cette hétérogénéité, de plus en plus d'approches d'intégration associent aux données des ontologies qui en définissent le sens. Ces approches sont dites « à base ontologique ».

Dans ce travail, nous proposons une approche et une architecteur d'un système d'intégration des sources de données relationnelles à base ontologique. Afin de valider notre approche nous implémentons un prototype de système d'intégration appelé HERO-Med (Higher Education Reference Ontology Mediator) appliqué sur des données universitaires.

Mots-clés : Intégration sémantique, hétérogénéité, approches médiateurs, entrepôt de données, ontologie, GaV, LaV.

Abstract

With the development of the Internet and intranets, the sharing of information from various distributed, autonomous and heterogeneous data sources becomes a critical need. In such a context, it is often necessary for an application to simultaneously access several data sources, because they contain relevant and complementary information. To do this, the solution of integration systems has been proposed. It consists to provide a transparent interface to relevant data via a global view. However, there are many types of heterogeneity and differences between the data sources that make data integration a difficult process. The semantic heterogeneity presents a major challenge; each may have a different perspective on the same concept. To reduce this heterogeneity, more and more integration approaches combine data with ontologies that define meaning. These approaches are called "ontology-based integration"

In this work, we propose ontology-based approach and system architecture for integrating relational data sources. To validate our proposed approach we implement a prototype of integration system called HERO-Med (Higher Education Reference Ontology Mediator) in the field of higher education and universities.

Keywords: Semantic integration, heterogeneity, mediators, data warehouse, ontology, GaV, LaV.

ملخص

نظرا لتطور شبكة الإنترنت والشبكات الداخلية، أصبح تبادل وتقاسم المعلومات القادمة من مصادر موزعة، مستقلة وغير متجانسة أمرا ضروريا. في هذا السياق، غالبا ما يكون ضروريا لتطبيق ما الوصول إلى مصادر عديدة في وقت واحد، ذلك أنها تحتوي على معلومات ذات صلة ومتكاملة. لهذا الغرض تم إقتراح حل تكامل أنظمة البيانات، الذي يوفير واجهة شفافة للبيانات ذات صلة عبر وجهة نظر موحدة. ومع ذلك، توجد أنواع عديدة من عدم التجانس والإختلاف بين مصادر البيانات التي تجعل من تكامل البيانات عملية صعبة. يعتبر التباين الدلالي تحديا كبيرا، لكل جهة أو نظام نظرة مختلفة على نفس المفهوم، للحد من هذا التباين وعدم التجانس، العديد من مناهج تكامل الأنظمة ترفق مع البيانات أنطولوجيا لتحدد المعنى. هذه المناهج تسمى مناهج على "أساس-الأنطولوجيا ".

من خلال هذا العمل، نقترح نهج قائم على الأنطولوجيا وبنية نظام لدمج مصادر البيانات العلائقية المتباينة، لكي نتحقق من صحة نهجنا المقترح قمنا بإنشاء نموذجا أوليا لنظام تكامل بيانات يسمى HERO-Med (الوسيط المرجعي لأنطولوجيا التعليم العالي) مطبق في مجال التعليم العالي والجامعات.

كلمات البحث : التكامل الدلالي، التباين، وسطاء، مستودع البيانات، الأنطولوجيا، LAV، GAV

Introduction Générale

« Plus s'étend et s'approfondie le champ de ma connaissance, plus s'aiguise la conscience de l'étendu de mon ignorance. »

Michel Beaud

Introduction générale

Sommaire

1 Problématique 2
2 Solution et approche 3
3 Organisation du mémoire 5

1. Problématique

Le problème de combiner des sources de données hétérogènes et de les interroger via une seule interface de requête ne date pas d'aujourd'hui. Depuis l'arrivée et l'adoption des bases de données dans les années soixante, l'accès transparent aux bases de données, leur partage et leur combinaison sont naturellement devenus indispensables et constituent un des challenges actuels de l'informatique.

L'expansion du nombre de sources d'information disponibles sur le Web et la quantité de données gérées au sein des bases de données des entreprises a rendu indispensable l'intégration de données provenant de différentes sources. Dans les domaines du e-sciences, e-gouvernement, e-learning, e-commerce, et de la bio-informatique ou encore des bibliothèques électroniques, des centaines de milliers de sources de données sont accessibles publiquement via le Web. Cette évolution quantitative des données et la diversité des formats des sources ont conduit les entreprises à consacrer 35 % de leur budget dédié aux technologies de l'information pour l'intégration de données. Ces sources de données sont le plus souvent réparties, autonomes et hétérogènes.

Un système d'intégration de données a pour rôle d'offrir à l'utilisateur une vue uniforme et une interrogation transparente des informations sans que l'utilisateur n'ait le souci de la provenance des informations ni de leur format d'origine.

L'intégration de données peut être effectuée de différentes façons et à différents niveaux de l'architecture du système. Deux approches principales pour la conception des systèmes d'intégration ont été définies en se basant sur la localisation des données gérées par le système :

- *l'intégration virtuelle de données*, où la vue unifiée est virtuelle et les données restent stockées dans les sources d'origine. L'architecture type pour l'intégration virtuelle de données est l'architecture médiateur ;
- *l'intégration matérialisée de données*, où la vue unifiée des données est matérialisée et les données sont rapatriées des sources d'origine et stockées dans un entrepôt de données.

Cependant, la quantité, la dispersion et la volatilité des ressources constituent autant de verrous que les systèmes d'intégration doivent lever. Les sources d'informations sont réparties : de plus en plus d'informations sont créées partout dans le monde et publiées sur le Web, de nombreuses entreprises ont des ramifications dans plusieurs pays et les états décentralisent leurs administrations. Elles sont autonomes car les sources de données sont conçues par différentes personnes, à différents moments et pour répondre à différents besoins applicatifs. Enfin, les sources d'informations sont hétérogènes : des logiciels différents sont utilisés pour créer et gérer les données (ex. Oracle, SQL Server, MySql, O2), les données sont publiées dans des formats divers (ex. HTML, PDF, images) et des modèles de données différents sont utilisés pour les représenter (ex. modèles relationnel, objet, semi-structuré).

Plusieurs types de conflits dus à l'hétérogénéité peuvent être considérés dans l'établissement des correspondances entre schémas lors de l'intégration de données. L'hétérogénéité provient des différents choix qui sont faits pour représenter les informations issues du monde réel dans un format informatique.

Elle se décline sous deux catégories :

- *Structurelle* : la manière dont sont représentées les données (exemple : l'adresse peut être représentée sur un seul champ, ou bien sur les champs : Rue, Code postal, Ville)
- *Sémantique* : en rapport avec la signification des données (synonymes, homonymes, etc.)

Nous nous intéressons particulièrement à l'intégration sémantique qui représente un défi aussi bien pour les chercheurs que pour les industriels.

Tout système d'intégration doit fournir les solutions aux problèmes suivants :

(1) Comment fournir une vue globale intégrée des données représentées à travers différentes conceptualisations ?

(2) Comment résoudre les problèmes d'hétérogénéité des sources de données ?

(3) Comment identifier et spécifier le mapping entre des données sémantiquement liées ?

L'état de l'art portant sur les systèmes d'intégration de données, présenté au premier chapitre est une contribution de ce mémoire. En effet, il englobe un état actualisé, mis à jour du contexte et souligné par la présenation d'une classification basée sur cinq critères orthogonaux des systèmes d'intégration.

2. Solution et Approche

L'intégration sémantique suscite de plus en plus d'intérêt avec l'émergence du Web sémantique. Grâce à ce dernier, les systèmes devraient pouvoir échanger des informations et des services dans un contexte sémantiquement riche. L'hétérogénéité sémantique est due aux multiples interprétations que des systèmes autonomes peuvent avoir de la même donnée. Ainsi, l'intégration de sources de données hétérogènes, autonomes et réparties passe par la résolution de ces conflits sémantiques et différences syntaxiques.

Afin de réduire cette hétérogénéité, de plus en plus d'approches d'intégration associent aux données des ontologies qui en définissent le sens. Ces approches sont dites *« à base ontologique »*. Deux catégories principales d'ontologies ont été utilisées dans l'élaboration de systèmes d'intégration : linguistiques ou formelles. Les ontologies linguistiques, visent à définir le sens des mots et les relations entre ces mots. Ces relations étant approximatives et fortement contextuelles elles permettent seulement une automatisation partielle du processus d'intégration, sous la supervision d'un expert humain.

Les ontologies formelles qui sont définies comme *une spécification explicite et formelle d'une conceptualisation commune*, par Thomas Gruber [81] peuvent contribuer à la résolution du problème de l'hétérogénéité sémantique. En effet, elles offrent une description formelle des concepts et de leurs relations existant dans certains domaines. Un utilisateur ou un concepteur peut regrouper et déclarer explicitement la sémantique des concepts et des données fournis par les sources. Ainsi, en exploitant cette sémantique explicite et partagée, l'impact nuisible de l'hétérogénéité sémantique peut être réduit. Il s'agit dans ce cas d'une approche d'intégration de données fondée sur les ontologies. Une fois que les données des sources sont rendues conformes à l'ontologie, les requêtes peuvent être exprimées en termes

de l'ontologie et les données peuvent être interrogées via une seule interface de requêtes. Dans le cas d'une intégration virtuelle des données ces requêtes sont réécrites en fonction des vues sur les sources et les extensions de ces vues sont ensuite intégrées et combinées pour pouvoir répondre à l'utilisateur. On note que, par la suite, on utilisera indifféremment les termes ontologie et schéma.

Il existe différentes utilisations possibles des ontologies dans un système d'intégration. Les approches les plus courantes sont des approches a posteriori. Lorsque des ontologies linguistiques sont utilisées, on part des mots du schéma. On se rapproche d'un thésaurus et on essaye de comprendre à travers les mots l'identité des concepts. Ceci sollicite une forte implication de l'expert et le résultat est très éventuel car les décisions sont subjectives.

Lorsque, des ontologies formelles sont utilisées, on suppose que chaque source contient sa propre ontologie. Le problème d'intégration sémantique devient alors un problème d'intégration d'ontologies. Comparée à la première méthode, la deuxième est :

1. Plus rigoureuse car les concepts sont définis avec plus de précision, l'appariement peut être plus argumenté,
2. Plus facilement outillées, en particulier, si les différentes ontologies sont basées sur le même modèle d'ontologie. Une correspondance entre ontologies peut alors être exploitée par des programmes génériques pour intégrer les données correspondantes.

Dans un processus d'intégration, les sources de données à intégrer peuvent être de nature structurée, semi-structurée ou bien non structurée. Dans notre approche les données sont de type structuré du fait qu'on s'intéresse à des bases de données relationnelles. L'approche proposée pour l'intégration de sources de données structurées hétérogènes, est fondée sur la médiation et les ontologies pour résoudre les conflits sémantiques et syntaxiques. Nous visons l'hétérogénéité sémantique aussi bien au niveau des schémas (hétérogénéités de schéma) qu'au niveau du contenu (hétérogénéités de valeur de données), et ceci, tout en préservant l'autonomie des sources de données. La résolution des conflits sémantiques entre les termes de la requête utilisateur et les termes des sources de données est le principal problème à traiter. Dans notre approche proposée pour l'intégration des sources de données :

- Nous utilisons une ontologie du domaine pour créer la requête de l'utilisateur. L'ontologie de domaine est mise au point par des experts du domaine et se compose de tous les termes et les concepts existants dans un domaine spécifique. Pour sa requête, l'utilisateur est limité à utiliser les termes de l'ontologie de domaine avec laquelle il doit être familier. Pour cela, nous proposons un modèle de représentation uniforme spécifique pour l'ontologie de domaine et une interface graphique adaptée à l'interaction de l'utilisateur afin d'exprimer des requêtes.
- Nous proposons un mécanisme de mapping entre les métadonnées des schémas locaux et le modèle de représentation uniforme de l'ontologie de domaine. Notre approche est de faire une correspondance entre ces structures et de stocker les informations de mapping dans un schéma relationnel. Le problème majeur dans les systèmes d'intégration de données à base ontologique est de détecter les similarités entre les schémas locaux et d'effectuer le mapping sémantique.

Notre système d'intégration proposé couvre les niveaux d'abstraction des conflits d'hétérogénéité entre les sources de données. Le système comprend :

- *Ontologie de domaine :* comme une solution pour résoudre les hétérogénéités des schémas.
- *Adaptateur :* comme solution pour résoudre les hétérogénéités des modèles de données.
- *Convertisseur :* solution pour résoudre les hétérogénéités de valeurs de données.

Afin de valider notre proposition, nous l'avons prototypée. Notre domaine d'application cible est celui de l'éducation supérieure et les universités. Il s'agit, de pouvoir intégrer automatiquement les sources de données hétérogènes de différentes universités à travers notre prototype, en offrant, sans effort de programmation, des possibilités d'accès ergonomiques. L'implémentation de notre prototype appelé HERO-Med (Higher Education Reference Ontology Mediator) est effectuée sur la Plateforme .NET en utilisant le langage ASP.Net dans Visual Studio 2012.

3. Organisation du mémoire

Ce mémoire est organisé autour de quatre chapitres :

Introduction générale.

1- Intégration des sources de données.

2- Technologie des ontologies et intégration des données.

3- Approche d'intégration à base ontologique proposée.

4- Implémentation et mise en œuvre de notre approche.

Conclusion et perspectives.

Le chapitre 1 présente un état de l'art portant sur le problème de l'intégration de sources de données réparties, autonomes et hétérogènes en détaillant les différents types de conflits. Dans ce chapitre nous définissons la notion d'un système d'intégration, ses différents composants, ses tâches principales et on montre les différentes phases du processus d'intégration des données hétérogènes. Nous présentons une classification intéressante des systèmes d'intégration basée sur cinq critères orthogonaux : (1) la représentation des données intégrées, (2) le sens de la mise en correspondance entre schéma global et schémas locaux, (3) la nature du processus d'intégration, (4) les langages de représentation de données intégrées, et (5) les niveaux d'abstraction. Avant de conclure ce chapitre, nous positionnons notre approche par rapport à cet état de l'art.

Le chapitre 2 est consacré à la présentation de la technologie des ontologies. Nous montrons leurs apports dans l'élaboration des systèmes d'intégration de données. Après une définition, nous présentons deux catégories de formalismes d'expression des ontologies : (1) les formalismes d'ontologie orientés gestion des données PLIB et RDFS et (2) les formalismes d'ontologie orientés inférences, où nous discutons en particulier du formalise OWL.

L'objet du chapitre 3 est de présenter notre approche et l'architecture de système d'intégration à base ontologique proposée. Cette approche nous permet l'intégration de sources de données hétérogènes structurées relationnelles. Elle est fondée sur la médiation et les ontologies de domaine pour résoudre les conflits sémantiques et syntaxiques. Nous visons l'hétérogénéité sémantique aussi bien au niveau des schémas (structures) qu'au niveau du contenu (données), et ceci, tout en préservant l'autonomie des sources de données. La résolution des conflits sémantiques entre les termes de la requête utilisateur et les termes de sources de données est le principal problème à traiter.

Le chapitre 4 présente la mise en œuvre et l'implémentation de prototype HERO-Med (Higher Education Reference Ontology Mediator) validant l'approche proposée. Notre domaine d'application cible est celui de d'éducation supérieure et les universités. Cette implémentation est effectuée sur la plateforme .NET en utilisant le langage ASP.Net dans Visual Studio 2012.

Enfin la conclusion générale de notre travail présente une esquisse des diverses perspectives.

Ce mémoire comporte trois annexes : l'annexe A présente le code en OWL de l'ontologie de domaine développée. L'annexe B donne les Scripts DDL (Data Definition Language) de création des sources de données entrant dans le scénario d'intégration. Enfin, l'annexe C présente le Code ASP.Net C# de l'implémentation de l'interface HERO-Med.

Chapitre 1
Intégration des sources de données

«Dans la vie, il n'y a pas de solution. Il y a des forces en marche : Il faut les créer et les solutions suivent.»

Antoine de Saint-Exupéry

Savoir pour bien Voir, Bien voir pour Comprendre, Et comprendre pour Savoir.

Chapitre I

Intégration des sources de données

Sommaire

1 Introduction 9
2 Problématique de l'intégration de données 10
2.1 Autonomie des sources 10
2.2 Interdépendances des sources 10
2.3 Hétérogénéités des sources 11
3 Systèmes d'intégration de données 13
3.1 Définition et Composante 13
3.2 Processus d'intégration 14
3.3 Tâche d'un système d'intégration 15
4 Classification des approches d'intégration 17
4.1 Architecture d'intégration / Localisation de données intégrées 17
4.1.1 Systèmes Multi-bases 17
4.1.2 Systèmes Fédérés 18
4.1.3 Architecture matérialisée (Entrepôts de Données) 18
4.1.4 Architecture virtuelle (Systèmes Médiateurs) 20
4.1.5 Architecture Mixte (Médiateurs / Entrepôt de données) 22
4.1.6 Systèmes P2P (Pair à Pair) 23
4.1.7 Discussion 23
4.2 Sens de mise en correspondance entre schéma global et schéma local 24
4.2.1 Global-as-View (GaV) 24
4.2.2 Local-as-View (LaV) 24
4.2.3 Generalized Local-as-View (GLaV) 25
4.2.4 BYU Global Local-as-View (BGLaV) 26
4.2.5 Both-as-View (BaV) 26
4.2.6 Discussion 26
4.3 Nature du processus d'intégration 27
4.3.1 Intégration manuelle de données 27
4.3.2 Intégration semi-automatique à l'aide d'ontologies linguistiques 27
4.3.3 Intégration automatique à l'aide d'ontologie conceptuelle 29
4.3.4 Discussion 31
4.4 Langages de représentation de données intégrées 31
4.4.1 Systèmes fondés sur un schéma global à base de règles 31
4.4.2 Systèmes fondés sur un schéma global à base de classes 31
4.4.3 Systèmes fondés sur un schéma global à base d'arbres 31
4.5 Niveaux d'abstraction 32
4.6 Comparaison entre les différentes approches d'intégration 32
5 Traitement des requêtes dans un système d'intégration 33
6 Principaux systèmes d'intégration 34
6.1 Systèmes d'intégration 34
6.2 Comparaison des principaux systèmes d'intégration 35
7 Positionnement de notre approche 37
8 Conclusion 38

1. Introduction

La réussite de l'Internet et l'Intranet a fait éclater le développement des sources d'informations sur le Web. Intégrer des données et des informations issues de sources autonomes, hétérogènes et évolutives est devenu un besoin essentiel pour un nombre important d'applications, comme les données d'une entreprise, le commerce électronique, le business intelligence (OLAP, fouille de données), la bio-informatique, etc. Cette intégration permet à ces applications d'exploiter cette mine d'information.

Pour réaliser l'intégration, une solution naïve suppose une connaissance fine des sources de données participant dans le processus d'intégration (les schémas de données, les contenus, les localisations, etc.). Cette solution a deux inconvénients majeurs : (1) le non-respect de la transparence d'accès aux données réparties et (2) un coût de mise en œuvre dans le cas d'un nombre important de sources. Une deuxième solution consiste à fournir aux utilisateurs une interface d'accès *unique*, *homogène* et *transparente* aux données de sources.

Plusieurs problèmes doivent être pris en compte pendant la conception de systèmes d'intégration. Ces problèmes proviennent de l'hétérogénéité structurelle et de l'hétérogénéité sémantique des données. L'hétérogénéité structurelle provient des différentes structures ou formats utilisés pour stocker les données de chaque source. L'hétérogénéité sémantique, par contre, présente un défi majeur dans le processus d'élaboration des systèmes d'intégration. Elle est due aux différentes interprétations des objets du monde réel.

L'objectif de ce chapitre est de bien positionner notre travail par rapport à l'état de l'art sur le problème de l'intégration de sources de données à base ontologique. Dans la section qui suit, nous présentons la problématique de l'intégration des données en détaillant les différents types de conflits. La section 3 définit la notion d'un système d'intégration, ses différents composants, ses principales tâches, par la suite nous montrons les différentes phases du processus d'intégration des données hétérogènes. Dans la section 4, nous présentons une classification des approches d'intégration en se basant sur les cinq critères orthogonaux suivants : (1) la représentation de données intégrées, (2) le sens de la mise en correspondance entre schéma global et schémas locaux, (3) la nature du processus d'intégration, (4) les langages de représentation des données intégrées, et (5) les niveaux d'abstraction de l'intégration. Cette classification va nous permettre de présenter de façon synthétique les méthodes d'intégration existantes. Elle nous permettra également de positionner précisément notre proposition d'intégration à base ontologique. La section 5 montre la façon dont les systèmes d'intégration traitent les requêtes. Dans la section 6 nous présentons les principaux systèmes d'intégration. Le positionnement de notre approche est décrit en section 7. Enfin, la section 8 conclut le chapitre.

2. Problématique de l'intégration de données

Avant d'aborder un état de l'art des systèmes d'intégration de données, il est important de définir la nature du problème à résoudre. Du fait du développement important de l'Internet, la recherche d'informations issues des sources de données réparties sur le réseau devient de plus en plus difficile [33]. En effet, grâce à la révolution des nouvelles technologies de l'information les entreprises aussi bien que les individus disposent d'une grande quantité de données. Ces données sont stockées dans des sources qui sont dans la plupart du temps hétérogènes et autonomes.

Par sources de données on entend, les informations enregistrées dans des bases de données traditionnelles et accessibles par des langages de requête très puissants, d'autres simplement stockées dans des systèmes de fichiers ou de simples tableurs, d'autres encore sont les résultats d'applications plus ou moins complexes, enfin, certaines sont les données variables et arborescentes des pages du Web. Ci-dessous nous décrivons trois problèmes fondamentaux soulevés lors de l'intégration des sources de données, qui provoquent d'autres défis importants pour l'intégration des données.

2.1 Autonomie des sources

Les sources de données qui participent à un scénario d'intégration sont autonomes. Cela signifie que les parties de l'organisation qui gèrent et contrôlent ces différentes sources de données sont indépendantes. Les administrateurs de sources de données permettent souvent à d'autres systèmes de partager des données sauf si le contrôle est conservé. En conséquence, pour construire un système d'intégration de données l'aspect de l'autonomie doit être considéré [1]. Les différents aspects de l'autonomie sont [18] :

- *Autonomie de communication :* chaque système local décide du moment où il communique avec le système d'intégration, c'est à dire quand il va répondre à une demande.
- *Autonomie d'exécution :* ce n'est pas la gestion du système d'intégration qui contrôle les transactions et les opérations locales ou externes des sources de données locales. En d'autres termes, une source de données locale dans un scénario d'intégration de données n'a pas besoin d'informer d'autres sources de son ordre d'exécution et de ses opérations.
- *Autonomie d'Association :* les systèmes locaux ont la possibilité de s'associer ou se dissocier du système d'intégration, c'est à dire qu'ils décident si, et comment, leurs fonctionnalités et leurs données participent au système d'intégration.

2.2 Interdépendance des sources

L'interdépendance implique que les données dans les différentes sources de données sont dépendantes les unes des autres. Différentes sources de données peuvent modéliser des univers de discours qui se chevauchent et contenir les mêmes objets du monde réel qui peuvent être en différents formats. Il peut aussi y avoir des dépendances entre les fonctionnalités des différentes sources de données. La gestion des données interdépendantes impose l'utilisation de contraintes de cohérence dans un système d'intégration.

2.3 Hétérogénéités des sources

L'un des problèmes les plus complexes lors de l'intégration de plusieurs sources de données autonomes est leur hétérogénéité. Chaque source de données est décrite par sa localisation, le type de données qu'elle gère, ses possibilités d'interrogation et le format des résultats.

- La localisation d'une source de données englobe tout aussi bien le référencement du site sur lequel se situe la source (URL, adresse IP + port, annuaire LDAP), que le protocole de communication utilisé (TCP/IP, IPX, Appletalk), les moyens d'accès à la base (ODBC, JDBC) ainsi que le support (pages Web, SGBD).
- Le type de données géré par une source peut être structuré (base de données relationnelles), semi structuré (XML,OEM:Object Exchange Model) ou non structuré (images, multimédia, texte libre).
- Les possibilités d'interrogation sont aussi nombreuses, et vont de langages de requêtes évolués et standardisés (SQL, OQL) ou propriétaires à de simples interfaces de programmation ou encore des recherches par motifs (moteur de recherche Web). Nous ferons abstraction des interfaces graphiques de requêtes utilisateurs, trop spécifiques et inadaptables dans le cadre d'une intégration.
- Enfin, les formats des résultats complètent les disparités qui peuvent exister entre les différentes sources de données. Celles-ci peuvent être formatées suivant divers standards (XML, HTML) ou encore accessibles par des structures de programmation variées (Result Set, OEM).

Les sources de données auxquelles nous avons accès dans un contexte interconnecté, via le Web, se sont diversifiées selon les directions que nous avons citées, de sorte que nous pouvons à présent parler, véritablement de, sources de données hétérogènes [2].

L'hétérogénéité provient des différents choix qui sont faits pour représenter les informations issues du monde réel dans un format informatique.

Elle se décline en deux catégories :

- ***Structurelle :*** la manière dont sont représentées les données (exemple : l'adresse peut être représentée sur un seul champ, ou bien sur les champs : Rue, Code postal, Ville).
- ***Sémantique* :** en rapport avec la signification des données (synonymes, homonymes, etc.)

La question fondamentale lorsque l'on veut faire inter-opérer des bases de données hétérogènes est d'une part, l'identification de conflits entre les concepts dans des sources différentes qui ont des liens sémantiques, d'autre part, la résolution des différences entre les concepts sémantiquement liés. Une taxonomie des conflits sémantiques a été proposée dans [6] : (1) conflits de représentations, (2) conflits de noms, (3) conflits de contextes, et (4) conflits de mesure de valeur que nous allons détailler dans les sections suivantes.

2.3.1 Conflits de représentation

Ces conflits se trouvent dans le cas où on utilise des propriétés différentes ou des schémas différents pour décrire le même concept. Cela signifie que le nombre de classes et propriétés représentant un concept dans les sources n'est pas le même.

2.3.2 Conflits de noms (termes)

Ces conflits se trouvent dans le cas où on utilise soit des noms différents pour le même concept ou propriété (*synonyme*), soit des noms identiques pour des concepts (et des propriétés) différents (*homonyme*).

2.3.3 Conflits de contextes

Le contexte est une notion très importante dans les systèmes d'information répartis. En effet, un même objet du monde réel peut être représenté dans les sources de données par plusieurs représentations selon un *contexte local* à chaque source. Ces conflits de contexte se trouvent dans le cas où les concepts semblent avoir la même signification, mais ils sont évalués dans différents contextes.

2.3.4 Conflits de mesure de valeur

Ces conflits sont liés à la manière de coder la valeur d'un concept du monde réel dans différents systèmes. Ils se trouvent dans le cas où on utilise des unités différentes pour mesurer la valeur de propriétés. Plusieurs autres types de conflits dus à l'hétérogénéité peuvent être considérés dans l'établissement des correspondances entre schémas lors de l'intégration de données. De nombreuses taxonomies de conflits, dont Sheth et Kashyap fournissent un large panorama [18], ont été proposées. Une version plus simple, constituée de six types de conflits, est reprise par Parent et Spaccapietra [8] (voir Tableau 1.1).

Tab. 1.1 – Taxonomie de conflits -Parent et Spaccapietra 96-[8].

Conflits	**Description**
Conflits de Classification	Les populations du monde réel représentées par les deux types sont différentes.
Conflits de Description	Les types ont des ensembles différents de propriétés et/ou leurs propriétés sont représentées de manière différente.
Conflits de Structure	Les concepts utilisés pour décrire les types sont différents.
Conflits d'Hétérogénéité	Les modèles de données utilisés sont différents.
Conflits de Méta données	Les conflits de ce type surviennent lorsqu'une donnée dans une base est en correspondance avec une Méta donnée (le nom d'un type) dans le schéma d'une autre base.
Conflits de Données	Des instances en correspondance ont des valeurs différentes pour des propriétés en correspondance.

3. Systèmes d'intégration de données

Les systèmes d'intégration de données offrent des architectures d'interopérabilité sur une fédération de sources de données distribuées, autonomes et hétérogènes. Les entrepôts de données, les systèmes de médiation et les architectures P2P sont des exemples d'infrastructures qui permettent l'intégration de données, c'est-à-dire l'accès à des données produites par des sources autonomes. A travers des schémas virtuels, des métadonnées et des correspondances sémantiques, ils permettent d'accéder à ces sources de données de façon uniforme et transparente, en transformant par réécriture les requêtes d'un utilisateur en sous-requêtes envoyées aux sources de données les plus appropriées. L'hétérogénéité des données extraites des sources nécessite leur réconciliation, en d'autres termes, leur mise en correspondance par rapport au schéma global, avant de les présenter à l'utilisateur.

3.1 Définition et Composante

Un système d'intégration de données fournit une vue unifiée de données provenant de sources multiples et hétérogènes. Il permet d'accéder à ces données à travers une interface uniforme, sans se soucier de leur structure ni de leur localisation [3].

Formellement, un système d'intégration de données est un triplet ***I* : <*G, S, M*>**, où : ***G*** représente le schéma global (défini sur un alphabet *AG*) modélisant le schéma intégré, ***S*** est l'ensemble des schémas des sources (défini sur un alphabet *AS*) décrivant la structure des sources participantes au processus d'intégration, ***M*** est une correspondance entre ***G*** et ***S*** qui établit la connexion entre les éléments du schéma global et ceux des sources.

Pour interroger le système intégré, les requêtes sont exprimées en termes de construction du schéma global ***G***. Par raisonnement, tout système d'intégration doit considérer l'intégration à deux niveaux :

- Le niveau schéma, qui consiste à consolider tous les schémas des sources en un seul schéma global, ou schéma de médiation, qui sera utilisé pour supporter les requêtes utilisateurs,
- Le niveau donné, qui consiste à extraire les instances à partir des sources de données.

Généralement, un système d'intégration comporte deux parties principales (voir Figure 1.1) : une *partie externe* et une *partie interne*.

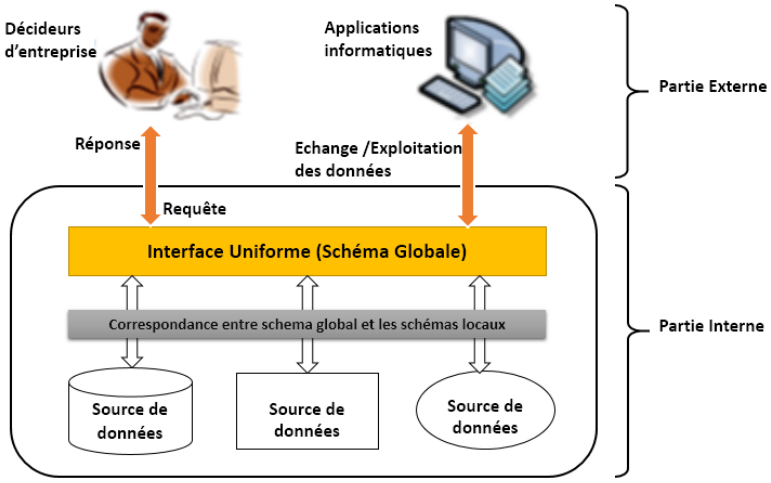


Fig. 1.1 – Système d'intégration de données.

1. **La partie externe** correspond aux utilisateurs du système d'intégration comme, par exemple, les décideurs d'une entreprise, ou des autres systèmes.
2. **La partie interne** comprend d'une part les sources de données qui participent dans le processus d'intégration et d'autre part une interface (schéma global) permettant aux éléments de la partie externe d'accéder d'une manière transparente aux sources de données. Cette interface peut être une couche sans données propres (*approche médiateur*) ou une couche contenant sous une forme qui lui est propre une duplication des données pertinentes des sources (*approche entrepôt*). Les éléments externes au système ne voient pas les sources internes. Celles-ci sont encapsulées, cachées par l'interface. Les éléments externes doivent pouvoir agir sur le système d'intégration d'une manière transparente via un schéma global.

3.2 Processus d'intégration

Étant donné un ensemble de sources de données hétérogènes *{S1, S2, ..., Sn}*, le problème d'intégration consiste à construire un schéma intégré (ou schéma global) qui sera utilisé comme interface d'accès aux sources de données. La construction du schéma global à partir des schémas locaux est une tâche difficile. Cette difficulté est liée au fait que les sources stockent différentes sortes de données, en différents formats, ayant différentes significations et associées aux différents noms [4].

Il convient d'abord d'indiquer qu'il existe plusieurs méthodologies permettant l'intégration de bases de données classiques. Une bonne revue des contributions sur le sujet peut être trouvée dans [5].

Les différentes phases du processus d'intégration des bases de données hétérogènes, sont : (1) Pré-intégration, (2) Comparaison, (3) Mise en conformité des schémas, (4) Fusion et Restructuration [7], [8] (voir Figure 1.2). Nous résumons ici les principales activités menées lors de ces phases.

- ***Pré-intégration :*** Cette phase commence par définir les langages et outils de modélisation, qui vont être utilisés dans la modélisation des contextes des différentes bases de données locales. Elle recommande également l'utilisation d'ontologies, de standards ou, en général, de formalismes qui devront être satisfaits par la solution d'intégration. Elle définit aussi les objectifs de l'intégration

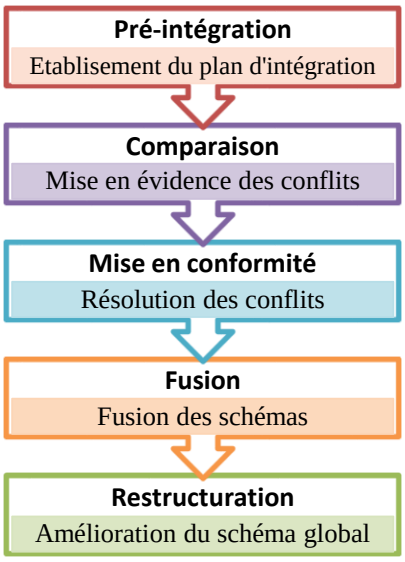


Fig. 1.2 – Processus d'intégration.

(Simplicité, complétude, exhaustivité), la stratégie d'intégration des différentes bases *(priorités, …)* et la typologie de la solution *(Multi-bases, fédération, médiation, entrepôt,…).* Trois facteurs, parfois liés au domaine d'application, ont une influence sur le choix du type d'intégration : la distribution, l'autonomie *(de communication, d'exécution, d'association),* et le degré d'hétérogénéité [9]. Ceux-ci déterminent la nature des composantes d'intégration et les modalités d'interaction entre celles-ci et les composantes locales.

- ***Comparaison :*** Dans cette phase, différents type de conflits seront mis en évidence *(conflits de désignation (homonymie, synonymie) structurels, de domaine, de contraintes….) cf.* section 2. on compare les schémas des bases de données existantes, en cherchant des portions de schémas qui soient intéressantes pour l'application cible. Ces portions de schémas peuvent être identiques dans les différentes bases, similaires ou bien concernent des concepts différents mais pouvant être rapprochés dans un but applicatif [8], [10].
- ***La mise en conformité des schémas :*** Cette phase traite les conflits induits par les correspondances, en suivant les objectifs fixés dans la phase de Pré-intégration. Ces conflits peuvent être de différents types, liés à la définition des entités, des structures, des domaines de valeurs, etc. Ils peuvent aussi porter sur des contraintes ; en effet, des contraintes existantes au niveau local peuvent générer implicitement des contraintes au niveau de la solution intégrée.
- ***La résolution des conflits :*** C'est une phase très délicate de l'intégration : la proposition de solutions conformes aux objectifs requiert parfois l'intervention d'experts des différents domaines.
- ***Fusion des schémas :*** Dans cette phase on produit les schémas des solutions intégrées, qui indiquent comment accéder aux données intégrées et comment les manipuler.
- ***Restructuration :*** Cette phase se retrouve plutôt dans les cas d'intégration de vues ou lorsque la modification ultérieure des schémas source est admise. Les approches récentes, de type fédérées, n'admettent pas de telles modifications, ceci afin de préserver l'environnement local (données, applications) des bases composant de la fédération et de garantir leur autonomie [11], cette étape a pour but l'amélioration du schéma global et la recherche de clarté sans remettre en cause la qualité.

3.3 Tâche d'un système d'intégration

On peut distinguer quatre tâches principales d'un système d'intégration. Les deux premières concernent la traduction de données provenant de sources différentes et résolvant le problème de l'hétérogénéité physique/logique des sources en fournissant une interface d'accès uniforme. Les deux dernières sont des tâches d'intégration sémantique et résolvent le problème de l'hétérogénéité sémantique en reliant chaque source au schéma global. Ces quatre tâches sont décrites ci-après :

- ***Transformation de données :*** Un exemple typique de cette tâche est la transformation de données relationnelles en XML et inversement. Les problèmes importants qui doivent être résolus à ce niveau sont la perte d'information, la taille des données générées et la performance des traitements sur ces données. L'exploitation de la structure de données à transformer (types, schémas) joue un rôle crucial dans ce contexte.
- ***Traduction de requêtes :*** La traduction de requêtes d'un langage (ex, XQuery) en un autre langage (ex, SQL) est liée au problème de transformation de données. Elle doit également prendre en compte la puissance d'expression du langage cible et nécessite souvent des extensions spécifiques afin d'obtenir la puissance d'expression du langage source.
- ***Réécriture de requêtes :*** Cette tâche est différente de la tâche précédente et généralement plus complexe car elle doit prendre en compte l'hétérogénéité structurelle et sémantique entre les schémas. Elle joue un rôle primordial dans l'intégration de données sur le Web.
- ***Fusion de données :*** La fusion de données essaye de répondre au problème de la représentation multiple d'une même information dans différentes sources. Elle fait partie de la tâche de réécriture de requêtes, mais se pose également dans un environnement où les données sont matérialisées.

Cette séparation fonctionnelle se traduit souvent par une séparation physique dans la forme d'un système d'intégration relié à un ensemble de traducteurs (adaptateurs). Néanmoins il est possible de déléguer des fonctions de traduction au système d'intégration ou, inversement, des fonctions d'intégration aux traducteurs. Par exemple, un traducteur plus intelligent peut préserver des informations structurelles sur les données sources afin de faciliter la réécriture et l'optimisation de requêtes. Inversement, le système d'intégration peut être conscient des langages de requêtes des sources et traduire directement les requêtes locales avant de les envoyer aux sources. Tous ces services font partie d'autres services plus généraux comme le traitement des requêtes et le stockage de métadonnées.

4. Classification des systèmes d'intégration

Plusieurs approches et systèmes d'intégration ont été proposés dans la littérature, souvent classifiés, à travers des critères différents [12], une vue générale est présentée dans [13]. Nous présentons une classification des systèmes d'intégration de données en se basant sur cinq critères orthogonaux :

1. Architecture d'intégration / Localisation de données intégrées.
2. Sens de mise en correspondance (Mapping).
3. Nature du processus d'intégration.
4. Langages de représentation de données intégrées.
5. Niveau d'abstractions de l'intégration.

4.1 Architecture d'intégration / Localisation de données intégrées

Ce critère spécifie si les données des sources locales sont dupliquées au niveau du système intégré ou pas. Les données du système intégré peuvent être (1) virtuelles (l'architecture médiateur) : le système intégré fournit alors une application chargée de jouer le rôle d'interface entre les bases de données locales et les applications des utilisateurs, ou (2) matérialisées (l'architecture d'un entrepôt de données) : les données issues des différentes sources de données sont dupliquées après modification éventuelle au sein du système [13].

4.1.1 Systèmes Multi-bases

Les systèmes Multi-bases sont des systèmes dits faiblement couplés [14]. On les caractérise de cette manière car ils n'offrent pas une vision unifiée des données. Il n'existe pas de schéma global permettant un accès transparent aux différentes sources de données. La coopération est seulement assurée par l'intermédiaire d'un langage commun : le langage Multi-base de type SQL notamment [15].

L'utilisateur peut poser une requête aux différents systèmes à l'aide de ce langage commun, sans se soucier de l'hétérogénéité des systèmes sources. Ceci ne signifie pas qu'une requête nécessitant l'accès à diverses sources est exécutée en une seule fois. L'utilisateur doit envoyer autant de requêtes qu'il y'a de sources impliquées. C'est donc à l'utilisateur de relier les différentes réponses aux requêtes formulées.

Ces systèmes sont donc faiblement intégrés et gardent une grande autonomie. Les sources peuvent évoluer de manière indépendante, sans conséquence sur leur accès. Cependant, la cohérence entre ces sources n'est pas assurée. L'hétérogénéité n'est pas traitée en amont. L'intégration est dynamique : les correspondances entre les données ne sont pas prédéfinies.

Certains systèmes offrent la possibilité de rendre les correspondances persistantes entre les sources par la création de vues Multi-bases. Les requêtes Multi-bases sont exprimées sur ces vues Multi-bases. C'est le cas du système MSQL [15].

4.1.2 Systèmes Fédérés

A l'inverse des systèmes Multi-bases, les systèmes fédérés sont dits fortement couplés [14]. Ils se caractérisent par l'existence d'un schéma unifié appelé schéma fédéré qui constitue l'interface d'accès au système d'intégration. L'intégration se situe au niveau des schémas.

La conception de ce schéma fédéré suppose d'unifier les schémas source et de traiter leur hétérogénéité. Il est nécessaire d'identifier les correspondances et de résoudre les conflits entre les éléments des schémas. Ces correspondances peuvent être exprimées à l'aide de différents langages, au moyen de règles ou à travers une ontologie. Il y'a donc cette fois une vision unifiée des sources (ou plus justement d'une partie). L'intégration offre un accès commun aux sources et une représentation commune.

L'intégration des systèmes fédérés est statique : les liens de correspondance entre les schémas sont prédéfinis. Ce n'est plus à l'utilisateur d'établir ces liens. Ce sont les administrateurs des bases sources qui définissent les sous-ensembles des données qu'ils souhaitent intégrer. L'accès aux données peut se faire de deux manières différentes : via les schémas locaux ou via le schéma fédéré. Les sources de données gardent leur autonomie et restent sous contrôle de leur administrateur. C'est une des particularités des systèmes fédérés. En général, seule une partie des données des différentes sources est mise en commun. Pour cette raison, ces systèmes sont parfois considérés comme faiblement couplés [8]. Le niveau d'intégration est en effet moins élevé que celui imposé par une base de données répartie ou distribuée. Cette dernière suppose une intégration totale des données sources. Les bases de données initiales perdent donc complètement leur autonomie. La fédération est un compromis entre une intégration nulle et totale [18].

4.1.3 Architecture Matérialisée (Entrepôts de Données)

Cette architecture consiste à centraliser physiquement, dans un *entrepôt de données*, l'ensemble de données consolidées à partir de diverses sources (catalogues électroniques, bases de données relationnelles, Web, etc.) (voir Figure 1.3).

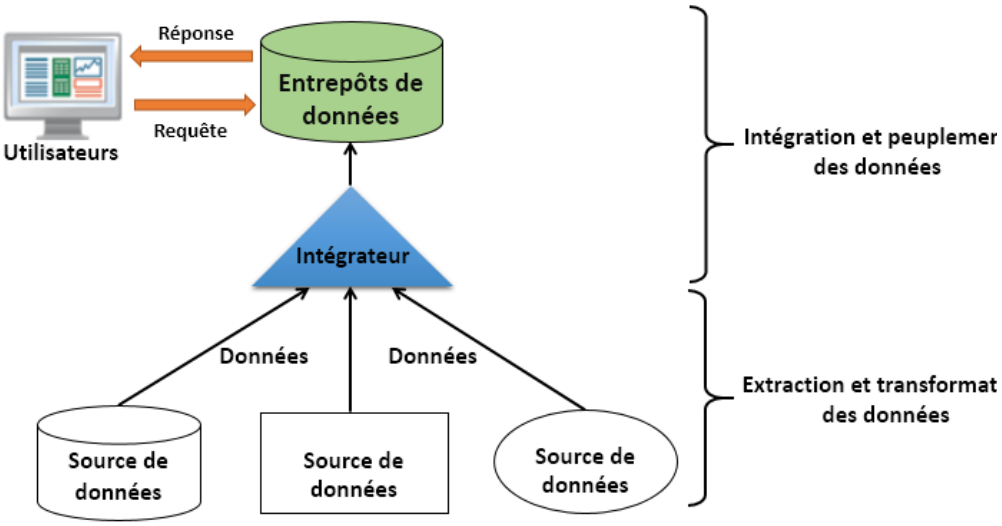


Fig. 1.3 – Intégration par l'entrepôt.

Giraldo [19] considère que la conception d'un système d'intégration selon une architecture matérialisée est similaire à celle des entrepôts de données. Le processus de construction d'un système d'intégration matérialisé se compose en quatre étapes principales [20] : (1) l'extraction des données des sources de données, (2) la transformation des données au niveau structurel, (3) l'intégration sémantique des données, et (4) le stockage des données intégrées dans le système cible. Ces étapes correspondent aux outils d'ETL (Extract, Transform and Load) [21].

Les deux premières étapes sont habituellement prises en charge par le même composant logiciel, appelé *adaptateur*. L'objectif d'un adaptateur est d'aboutir à des données représentées dans un même format. Chaque adaptateur fournit ainsi une interface d'accès et de requêtes sur la source. Les données extraites sont ensuite intégrées en éliminant les conflits, puis stockées dans l'entrepôt.

L'avantage principal d'une telle approche est que l'interrogation d'un entrepôt de données se fait directement sur les données de l'entrepôt et non sur les sources originales. On peut donc utiliser les techniques d'interrogation et d'optimisation des bases de données traditionnelles. Par contre cette approche exige un coût de stockage supplémentaire et surtout un coût de maintenance causé par les opérations de mises à jour au niveau de sources de données (toute modification dans les sources locales doit être répercutée sur l'entrepôt de données). Le problème de maintenance du système d'intégration matérialisée est similaire au problème de maintenance des vues matérialisées [22].

Cette approche est mieux appropriée dans le cas où des entreprises veulent stocker localement toutes les données de diverses sources au sein d'une base de données d'entreprise unique pour des raisons privées.

Quelques projets spécifiques d'entrepôts de données servent actuellement de références, en particulier, le projet européen DWQ (Data Warehouse Quality) [23] et le projet WHIPS (WareHouse Information Prototype at Stanford) [24, 25]. Le but du WHIPS est de développer des algorithmes pour collecter, intégrer et maintenir des informations émanant de diverses sources. Une architecture a été développée ainsi qu'un Prototype pour identifier les changements de données des sources, les transformer et les synthétiser selon les spécifications de l'entrepôt, et les intégrer de façon incrémentale dans l'entrepôt. La Figure1.4 présente les modules principaux du WHIPS [25] :

- Les données de l'entrepôt sont représentées selon le modèle relationnel : les vues sont définies dans ce modèle et l'entrepôt stocke les relations. Ces vues sont formulées par l'administrateur à travers le *spécificateur de vue*.
- Chaque vue d'entrepôt est associée à une *gestion de vue*. La gestion de vue est utilisée pour synchroniser des modifications survenues à cette vue en même temps. Pour chaque modification effectuée, une requête de mise à jour est créée et envoyée au *processeur de requête*.
- Le *moniteur* est chargé de détecter des modifications des sources. Chaque modification d'une source sera signalée à l'*intégrateur* pour analyser son influence sur les vues matérialisées dans l'entrepôt.

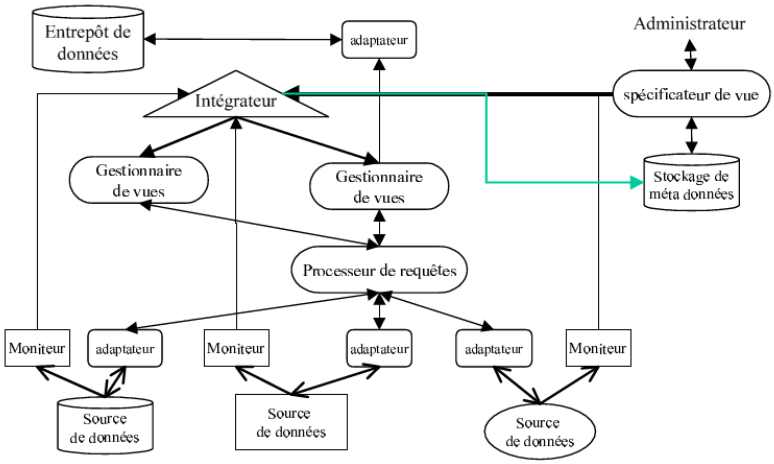


Fig. 1.4 – Architecture du système WHIPS pour la maintenance de l'entrepôt de données.

– Les *adaptateurs*, quant à eux, traduisent les requêtes exprimées sous forme d'une représentation relationnelle en des requêtes dans le langage natif de la source. L'utilisation d'un adaptateur par source cache les détails d'interrogation (spécifique à la source) au processeur de requêtes. L'adaptateur de l'entrepôt reçoit les définitions de vues et les modifications sur les données des vues dans un format canonique, et les traduit dans la syntaxe spécifique du système de gestion de base de données de l'entrepôt. Tous les adaptateurs supportent la même interface de requêtes bien que leur code interne dépend de leur source.

– L'*intégrateur* a pour rôle principal de faciliter la maintenance des vues en calculant les modifications des sources qui ont besoin d'être propagées dans les vues. Ces modifications sont ensuite transférées aux *gestionnaires de vues*.

– Le *processeur de requêtes* reçoit les requêtes de mise à jour des *gestionnaires de vues* et génère les requêtes appropriées aux adaptateurs de chaque source.

4.1.4 Architecture Virtuelle (Systèmes Médiateurs)

Cette architecture consiste à développer une application chargée de jouer le rôle d'interface entre les sources de données locales et les applications d'utilisateurs. Ce type d'approche a été utilisé par un nombre important de projets [26, 27, 28, 29, 64]. Il repose sur deux composants essentiels : le *médiateur* et l'*adaptateur* (voir Figure1.5).

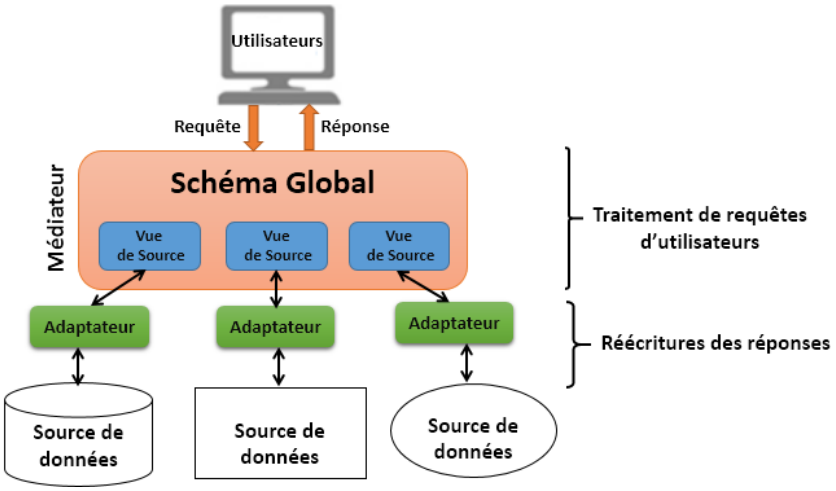


Fig. 1.5 – Intégration par le médiateur.

- ***Le médiateur :*** chargé de la localisation des sources de données et des données pertinentes par rapport à une requête, il résout de manière transparente les conflits de données. Un ensemble de connaissances sur les sources permet au médiateur de générer un plan d'exécution pour traiter les requêtes d'utilisateurs ;
- ***L'adaptateur :*** comme dans l'approche entrepôt, c'est un outil permettant à un (ou plusieurs) médiateur(s) d'accéder au contenu des sources d'informations dans un langage uniforme. Il fait le lien entre la représentation locale des informations et leur représentation dans le modèle de médiation.

Dans cette approche, les données ne sont pas stockées au niveau du médiateur. Elles restent dans les sources de données et ne sont accessibles qu'à ce niveau. L'intégration des données est fondée sur l'exploitation de vues abstraites décrivant de façon homogène et uniforme le contenu des sources d'information dans les termes du médiateur. Les sources d'information pertinentes, pour répondre à une requête, sont calculées en fonction de la définition de ces vues. Le problème consiste à trouver une requête qui est équivalente à celle de l'utilisateur. Les réponses à la requête posée sont ensuite obtenues en évaluant cette requête sur les extensions des vues. Le problème d'hétérogénéité sémantique entre les différentes sources d'information ne se pose pas pour l'utilisateur puisqu'il est résolu par le médiateur. L'approche médiateur présente l'intérêt de pouvoir construire un système d'interrogation de sources de données sans toucher aux données qui restent stockées dans leur source d'origine. Par contre le médiateur ne peut pas évaluer directement les requêtes qui lui sont posées car il ne contient pas de données, ces dernières étant stockées de façon distribuée dans des sources indépendantes. Dans cette approche, deux problèmes apparaissent : (1) comment construire le schéma intégré ? et, (2) comment effectuer la mise en correspondance entre les termes utilisés dans ce schéma et les structures des schémas des sources qui contiennent les données ?.

De nombreux travaux se sont portés sur l'architecture virtuelle. Hacid et al. [20] ont identifié trois grands problèmes ayant fait l'objet d'études sur. (1) Les langages pour modéliser le schéma global, afin de représenter les vues sur les sources à intégrer et pour exprimer les requêtes provenant des utilisateurs humains ou d'entités informatiques. (2) La conception et la mise en œuvre d'algorithmes de réécriture de requêtes en termes de vues décrivant les sources de données pertinentes ; et (3) la conception d'interfaces intelligentes assistant l'utilisateur dans la formulation de requêtes, l'aidant à affiner une requête en cas d'absence de réponses ou en cas de réponses beaucoup trop nombreuses.

Un objet **OEM** est structuré comme : < oid, étiquette, type, valeur >, où :
- **oid :** un identifiant,
- **étiquette :** une chaîne de caractères qui décrit ce que l'objet représente,
- **type :** le type de la valeur de l'objet,
- **valeur :** la valeur de l'objet.

Description d'un disque dur :

< o1, 'DisqueDur', set, {o2, o3, o4, o5} >
< o2, 'Situation', boolean, true >
< o3, 'Marque', string, 'DELL' >
< o4, 'Garantie', integer, 3 >
< o5, 'Capacité'>, integer, 80 >

Fig. 1.6 – Exemple de données OEM.

Nous présentons dans la Figure 1.6 un exemple du système médiateur : le projet TSIMMIS (The Stanford-IBM Manager of Multiple information Source) [26] de l'Université de Stanford qui est considéré comme un des premiers projets d'intégration de données appliquant cette architecture. TSIMMIS a pour objectif de produire des outils pour l'accès intégré à des sources d'information. Il utilise une hiérarchie de médiateurs pour intégrer des sources de données hétérogènes. Le modèle commun utilisé dans TSIMMIS est OEM (Object Exchange Model) [34]. Les données OEM sont représentées par un graphe, dont les nœuds sont des objets permettant de stocker les données et leurs schémas. Chaque source d'information est équipée d'un adaptateur qui encapsule la source, en convertissant les objets de données en OEM. Chaque médiateur obtient d'abord les informations (sous forme d'OEM) de plusieurs adaptateurs ou d'autres médiateurs. Il traite ensuite ces informations en intégrant et en résolvant les conflits entre les fragments d'information des différentes sources, et enfin fournit l'information résultante à l'utilisateur ou aux autres médiateurs. Les médiateurs sont traités comme des vues sur les données trouvées dans les sources qui sont convenablement intégrées et traitées. Le médiateur est défini en termes d'un langage logique appelé MSL. MSL est essentiellement un DATALOG étendu pour supporter les objets OEM. Les médiateurs réalisent typiquement des vues virtuelles puisqu'ils ne stockent pas les données localement. Quand un utilisateur pose une question au système, un médiateur concernant cette question est sélectionné. Ce médiateur décompose la requête et puis, propage les sous-requêtes. Lorsqu'une nouvelle source est ajoutée, TSIMMIS nécessite de changer tous les médiateurs qui l'utilisent. C'est précisément ce problème qui peut être évité en changeant le sens de définition des vues.

4.1.5 Architecture Mixte (Médiateurs / Entrepôt de données)

Avec le développement du Web et des technologies de l'information ces dernières années, d'autres approches d'intégration tels que les systèmes hybrides (approche mixte) ont été proposés. Ces systèmes combinent à la fois l'approche médiateur et l'approche entrepôt. Il s'agit, par exemple, d'un système médiateur qui intègre plusieurs sources de données externes et qui exploite un entrepôt de données contenant des données conformes au schéma global du médiateur. XYLEME [49] est un exemple de ce type d'architecture. Il adopte l'approche mixte dans son architecture. Les ontologies globales et locales sont exprimées à l'aide d'arbres, avec un mapping GaV/LaV. Le mapping est réalisé semi automatiquement par

la génération de tables de correspondance entre les chemins de l'ontologie globale et les chemins des ontologies locales.

Un autre exemple qui suit l'architecture de médiation Hybride est PICSEL 2 [64]. Le médiateur intègre également un entrepôt de données local qui contient des données fournies par les fournisseurs de contenu. Ces données sont conformes au schéma de médiateur représenté dans le langage RDFS + (RDFS *« Resource Description Framework Schema »* prolongé par un fragment de OWL DL *« Ontology Web Language/ Description Logics »*). L'entrepôt de données sera progressivement enrichi par des données externes.

4.1.6 Systèmes P2P (Pair à Pair)

L'émergence de systèmes pair à pair (Peer-to-Peer) de partage de fichiers a conduit les chercheurs à considérer l'architecture P2P dans le contexte de l'intégration et le partage de données [35][36][37][38][39]. Ces systèmes d'intégration P2P suivent une approche décentralisée pour l'intégration de pairs autonomes et distribués contenant des données qui peuvent être partagées. L'objectif principal de tels systèmes est de fournir une interopérabilité sémantique entre plusieurs sources de données en l'absence de schéma global. Chaque pair est interconnecté avec un certain nombre de pairs du réseau (appelés voisins) à l'aide de formules de coordination [41]. Edutella [37] [40], SomeWhere [36], PIAZZA [42] sont des exemples de travaux récents sur les systèmes P2P.

4.1.7 Discussion

Nous venons d'exposer brièvement différents systèmes d'intégration, des architectures qu'il est possible de mettre en place pour faire coopérer de manière relativement transparente, selon le niveau d'intégration des bases de données initialement indépendantes. Nous avons ainsi vu les systèmes Multi-bases, les systèmes fédérés et les systèmes de médiation. Nous avons également présenté les entrepôts de données car il existe une phase d'intégration de données pour les constituer.

Dans le cadre de notre travail, nous irons jusqu'au développement d'une architecture d'un système intégré mais le problème que nous avons traité s'inscrit dans un processus d'intégration destiné à concevoir un système intégré de sources de données à base ontologique. La médiation est souhaitée car l'intégration dans notre contexte est orientée vers l'interrogation de plusieurs sources. Elle doit être d'un niveau assez élevé (une représentation unifiée doit être fournie) et l'accès aux données doit être possible en lecture et écriture. Par ailleurs, les bases sources doivent garder une certaine autonomie car elles correspondent chacune à des données spécifiques. Ces données doivent continuer d'exister indépendamment de la médiation. Ajoutons encore que la médiation est bien adaptée car l'intégration doit être dynamique.

4.2 Sens de mise en correspondance entre schéma global et schémas locaux

Ce critère permet de définir les relations qui existent entre le schéma global et les schémas locaux. Deux principales approches ont été proposées : GaV (Global-as-View) et LaV (Local-as-View). Des approches mixtes sont développées par la suite comme : GLaV (Generalized Local-as-View), BGLaV (Global Local-as-View) et BaV (Both-as-View).

4.2.1 Global-as-View (GaV)

L'approche GaV a été la première à être proposée pour intégrer des informations. Elle consiste à définir à la main (ou de façon semi-automatique) le schéma global en fonction des schémas des sources de données à intégrer puis à le connecter aux différentes sources. Pour cela, les prédicats du schéma global, aussi appelés relations globales, sont définis comme des vues sur les prédicats des schémas des sources à intégrer. Comme les requêtes d'un utilisateur s'expriment en termes des prédicats du schéma global, on obtient facilement une requête en termes de schéma des sources de données intégrées, en remplaçant les prédicats du schéma global par leurs définitions.

Parmi les systèmes utilisant GaV, on peut citer TSIMMIS [26] et MOMIS [43]. En effet, dans TSIMMIS, les médiateurs sont conceptuellement représentés par des vues définies sur une ou plusieurs sources. Dans le cas d'ajout d'une nouvelle source, TSIMMIS reconstruit les médiateurs.

4.2.2 Local-as-View (LaV)

LaV est l'approche duale, car elle suppose l'existence d'un schéma global et elle consiste à définir les schémas des sources de données à intégrer comme des vues du schéma global.

Les principaux systèmes développés autour de cette approche sont : InfoMaster [27], PICSEL [64], Information Manifold [28].

InfoMaster [27], par exemple, est un entrepôt virtuel de catalogues développé par l'Université de Stanford (voir Figure 1.7). Il présente à l'utilisateur un catalogue virtuel, appelé *schéma référencé*,

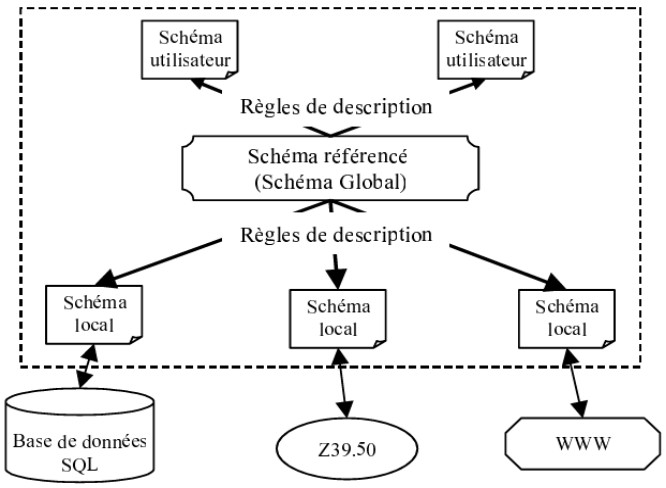


Fig. 1.7 – Architecture du système InfoMaster.

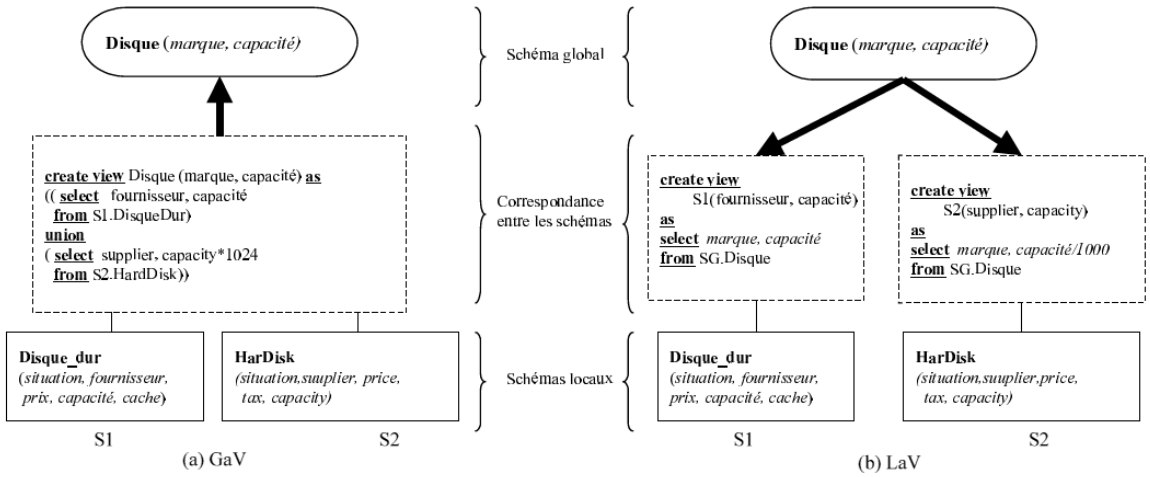


Fig. 1.8 – Exemple du sens de mise en correspondance entre schéma global et schéma local [33].

sur lequel il formule ses requêtes. Ce système utilise le langage KIF [1] (Knowledge Interchange Format) pour définir une liste de règles qui associe les schémas des sources intégrées aux vues du *schéma référencé*. Ces règles permettent aussi de convertir l'information afin de la mettre sous format adéquat et de l'adapter aux sources.

- **Bilan sur les approches GaV et LaV**

On considère souvent que l'adjonction d'une nouvelle source est plus facile dans l'approche LaV que dans l'approche GaV [20]. En fait, cela dépend de l'hypothèse faite concernant (1) les concepts contenus dans le schéma global, et (2) les concepts existants (auxquels le système d'intégration doit donner accès) dans la nouvelle source. Si l'on fait l'hypothèse qu'une nouvelle source ne doit pas modifier le schéma global (hypothèse faite dans les systèmes LaV), soit qu'elle ne contient aucun nouveau concept, soit comme présenté dans la Figure 1.8, que le schéma global ne représente que partiellement les différentes sources, l'effet d'adjonction est tout à fait similaire. Le coût de modification de l'implémentation de la vue globale, dans l'approche GaV, est contre balancée, dans l'approche LaV, par le coût de définition du schéma local comme vue du schéma global, puis par le coût de réécriture de requêtes en fonction des vues.

4.2.3 Generalized Local-as-View (GLaV)

Les avantages des approches LaV et GaV ont été combinés dans des approches mixtes. C'est ainsi qu'a été proposée l'approche GLaV (Generalized Local-as-View) [46], qui utilise des règles de correspondance ayant plus d'un terme dans leur entête (l'entête peut être la combinaison d'un nombre quelconque de prédicats, contrairement à l'approche LaV ou GaV).

Dans GLaV (Generalized Local-as-View), on dispose de vues au niveau global et local. Une correspondance entre les vues au niveau global et local est requise. Ces correspondances n'ont pas de direction et s'appliquent dans les deux sens puisque les concepts dans l'ontologie globale sont considérés comme des vues. En fait, l'approche Hybride permet d'avoir la correspondance entre les ontologies locales via le vocabulaire partagé. Donc la correspondance peut être calculée via l'ontologie globale. Dans le cas de l'approche Hybride modélisée selon GLaV, nous gardons une indépendance entre les sources, et nous pouvons calculer indirectement les correspondances entre les sources. Cette caractéristique est impérative si nous voulons avoir des résultats cohérents.

[1] http://www-ksl.stanford.edu/knowledge-sharing/kif/

4.2.4 BYU Global Local-as-View (BGLaV)

Cette approche est venue juste après l'approche GLaV, L'approche BGLaV (Brigham Young University Global-Local-as-View) se présente comme un point de vue alternatif qui n'est ni GaV ni LaV. L'approche emploie des mapping de la source à la source cible basés sur un schéma conceptuel prédéfini de la source cible, qui est spécifié ontologiquement et indépendamment des sources les unes des autres. Il est plus facile de maintenir le système d'intégration proposé avec BGLaV contrairement à GaV et LaV, la reformulation de requêtes est réduite au déploiement de règles. Comparée à d'autres approches d'intégration de données, BGLaV combine les avantages de GaV et LaV, réduit leurs inconvénients, et fournit une alternative pour l'intégration de données flexibles et extensibles.

Lors de l'ajout d'une nouvelle source, le système d'intégration ajoute des nouvelles règles représentant la vue de la nouvelle source.

4.2.5 Both-as-View (BaV)

L'approche BaV (Both-as-View) [44], utilise des transformations incrémentales afin de lier deux schémas. L'approche BaV a été introduite dans le cadre du projet d'intégration AutoMed [45].

4.2.6 Discussion

Dans les systèmes d'intégration basés sur les ontologies et relativement à l'approche Hybride, le schéma global et les schémas locaux sont des ontologies. Trois modèles d'intégration peuvent être appliqués à cette approche : GaV, LaV, GLaV. Les systèmes XYLEME [49] et PICSEL [64] utilisent l'approche Hybride. BGLaV et BaV sont aussi des approches hybrides.

Razor [47], Internet Softbot [48], InfoMaster [27], Information Manifold [28] sont des systèmes fondés sur un schéma global à base de règles, suivant tous l'approche LaV. Le système HERMES [74], suit une approche GaV. Les systèmes fondés sur un schéma global à base de classes, tel que TSIMMIS [26] et MOMIS [43] suivent l'approche GaV, à l'inverse de KRAFT [32] et OBSERVER [29] suivent LaV. Le système PICSEL [64], suit une approche LaV, XYLEME [49] et C-Web [50], relèvent à la fois des deux approches GaV et LaV et sont des systèmes fondés sur un schéma global à base d'arbres.

BGLaV, est une approche d'intégration combinant les avantages et évitant les inconvénients de GaV et de LaV. Elle augmente l'évolutivité et la rentabilité au-dessus des approches précédemment proposées.

4.3 Nature du processus d'intégration

Ce critère spécifie si le processus d'intégration de données est effectué d'une manière manuelle, semi-automatique ou automatique. Ce critère devient essentiel lorsque l'on veut intégrer un nombre important de sources de données indépendantes.

4.3.1 Intégration manuelle de données

Les méthodes d'intégration manuelle de données correspondent à la première génération de systèmes d'intégration. Cette génération est apparue dans les années 90 [23, 24, 25, 26]. A cette époque, les travaux d'intégration de données visaient essentiellement à automatiser l'interopérabilité syntaxique de données. Par contre, les conflits sémantiques étaient résolus d'une façon manuelle. Puisque seul un observateur humain pouvait interpréter la sémantique des données, ces approches laissaient à l'administrateur (ou l'utilisateur) la responsabilité du processus d'intégration.

D'après Parent et al. [8], les approches d'intégration manuelle visent à fournir des langages de manipulation de schémas que l'administrateur ou l'utilisateur peut utiliser pour construire lui-même (si le langage est procédural) ou spécifier (si le langage est déclaratif) le schéma intégré. Les langages procéduraux offrent des primitives de transformation de schéma qui permettent de restructurer les schémas initiaux jusqu'à ce qu'ils puissent être fusionnés en un seul schéma. Le système génère alors automatiquement les règles de traduction entre les schémas initiaux et le schéma intégré. Les langages déclaratifs sont plus faciles à utiliser, mais l'établissement des règles de traduction par le système est plus délicat. Les stratégies manuelles supposent la connaissance par l'administrateur de la structure du schéma intégré.

Nous citons ici trois méthodes typiques. La première, nommée *système de Multi-bases de données* [55], qui vise à construire un système d'accès à de multiples bases de données. La deuxième, nommée la *fédération des bases de données* [18], et consiste à fédérer les bases de données intégrées. La troisième consiste à intégrer les données en utilisant des standards de représentation ou de formalisation comme : Corba/IDL [57] [2], XML [3], ou *EXPRESS* [54].

Dans les environnements qui nécessitent d'intégrer un nombre important de sources de données réparties qui évoluent fréquemment, l'intégration de données manuelle devient très coûteuse et souvent même impossible. Des traitements plus automatisés ont donc été conçus pour faciliter la résolution des conflits sémantiques. Ceci correspond aux deux classes étudiées ci-dessous.

4.3.2 Intégration semi-automatique à l'aide d'ontologies linguistiques (ou thésaurus)

Afin d'atteindre une intégration au moins partiellement automatique de différentes sources de données, plusieurs groupes de recherche ont développé des techniques d'intégration basées sur des notions d'ontologies, issues de la communauté de l'intelligence artificielle. Le rôle d'une ontologie est de servir de pivot pour définir la sémantique des différentes données à l'aide de concepts communs et formalisés, compréhensibles et admis par tous les participants (utilisateurs).

La deuxième génération de systèmes d'intégration utilise des ontologies linguistiques (*OL*). Une OL définit

[2] http://www.omg.org/gettingstarted/omg_idl.htm
[3] http://www.w3.org/XML/1999/XML-in-10-points.html

le sens des mots utilisés dans un domaine d'´étude. Ces mots sont essentiellement liés par des relations linguistiques telles que la synonymie, l'antinomie, l'hyperonymie, etc. Les OL, aussi appelées des thésaurus, sont utilisées pour identifier automatiquement ou semi-automatiquement quelques relations sémantiques entre les termes utilisés dans les sources de données.

On peut alors automatiquement comparer les noms de relations ou d'attributs et essayer d'identifier les éléments similaires en utilisant des *OL*. Les *OL* restent cependant orientées « terme » et non « concept ». Cela pose notamment des problèmes d'homonymie et de dépendance vis-à-vis d'un langage. Ce type d'ontologies est également très lourd à développer (dizaine de milliers de concepts) et à mettre à jour. Enfin, les relations entre termes sont très contextuelles. Ainsi le traitement par *OL* est nécessairement supervisé par un expert et ne peut être que partiellement automatique. Certains travaux d'intégration à base d'*OL* utilisent des mesures d'affinité et de similarité afin de calculer la vraisemblance de relations entre concepts [31, 22, 32]. Ces mesures sont associées aux seuils définis par l'administrateur de la base de données utilisés pour décider si la relation est retenue. De telles procédures compromettent la qualité du système d'intégration. C'est pourquoi ce type d'approche ne peut être complètement automatique que pour l'interprétation, automatique, mais approximative de documents.

Deux thésaurus assez connus sont utilisées dans ce genre d'approche : WordNet et MeSH (Medical Subject Heading) [4]. MeSH est un thésaurus médical. C'est le thésaurus d'indexation de la base bibliographique MEDLINE. Le MeSH offre une organisation hiérarchique et associative et comprend jusqu'à neuf niveaux de profondeur. L'étude de MeSH montre que l'on est en face d'un thésaurus développé pour l'indexation et non pour les inférences [60]. WordNet [61] est une base de données lexicale. Les termes y sont organisés sous formes d'ensembles de synonymes, les *synsets*. Chaque *synset* est un concept lexicalisé. Ces concepts lexicalisés sont reliés par des relations linguistiques. WordNet est un énorme dictionnaire hypermédia de l'anglais-américain (plus de 100 000 *synsets*). Sa richesse et sa facilité d'accès le positionnent comme un intéressant outil pour la recherche d'information ou d'autres tâches comme le traitement du langage naturel mais ce n'est pas une ontologie car les relations ne sont en aucun cas formelles. L'utiliser tel quel, dans un système formel est donc voué à l'échec. Sa seule utilisation dans le cadre de l'intégration ne peut donc être que d'assister un expert humain.

Le projet MOMIS (Mediator envirOnment for Multiple Information Sources) [43] est un exemple de projet d'intégration utilisant des *OL* qui vise à intégrer semi automatiquement des données de sources structurées et semi structurées. Pour cela, chaque source de données est associée à un adaptateur qui consiste à traduire le schéma de cette source dans un modèle orienté objet commun (*ODLi*[3]) [62]. Un thésaurus global représentant les relations terminologiques entre les schémas des sources est d'abord construit, à l'aide de WordNet, en extrayant les termes utilisés dans les schémas des sources.

[4] http://mesh.inserm.fr/mesh/

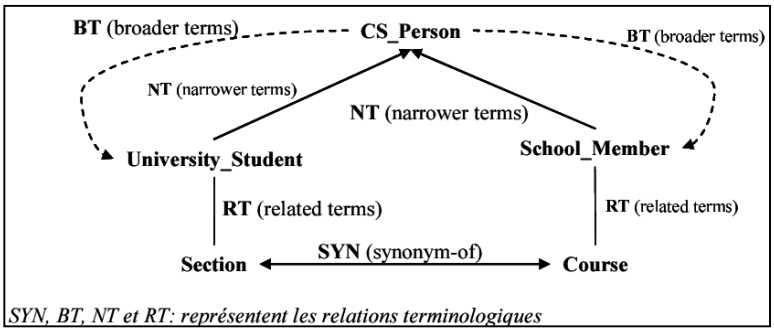


Fig. 1.9 – Un exemple de thésaurus [58].

Une vue intégrée est ensuite créée semi automatiquement en se basant sur le thésaurus et les mesures d'affinité de concepts. Ces dernières calculent d'abord les similitudes entre deux concepts : affinité de nom, affinité structurelle, et affinité globale. Puis, le résultat de ce calcul est comparé avec un *seuil pré-choisi* afin d'introduire les concepts plus généraux correspondant à la vue intégrée. Les concepts introduits dans la vue intégrée sont des subsumants communs à plusieurs termes apparus dans plusieurs schémas. La Figure1.9 montre l'exemple d'un thésaurus global [58]. Ce dernier est représenté sous forme d'un graphe orienté et étiqueté:

- les nœuds représentent les termes du thésaurus (*CS Person, University Student, School Member, Section, Course*),
- les arcs représentent les relations terminologiques entre les termes,
- les étiquettes définissent la nature des relations terminologiques comme : SYN (synonym), BT (broader terms), NT (narrower terms) et RT (related terms).

4.3.3 Intégration automatique à l'aide d'ontologie conceptuelle

La troisième génération des systèmes d'intégration consiste à associer aux données une ontologie conceptuelle (*OC*) qui en définit le sens. Une ontologie conceptuelle est "*une spécification explicite et formelle d'une conceptualisation faisant l'objet d'un consensus*" [63]. En fait, dans une conceptualisation, le monde réel est appréhendé à travers des concepts représentés par des classes et des propriétés. Des mots d'un langage naturel peuvent être associés, mais ce ne sont pas eux qui définissent le sens des concepts. C'est l'ensemble des caractéristiques associées à un concept ainsi que ses liens avec les autres concepts qui en définissent le sens. Une *OC* regroupe ainsi les définitions d'un ensemble structuré de concepts. Celles-ci sont traitables par machine et partagées par les utilisateurs du système. Elles doivent, en plus, être explicites, c'est-à-dire que toute la connaissance nécessaire à leur compréhension doit être spécifiée.

La référence à une telle ontologie est alors utilisée pour éliminer automatiquement les conflits sémantiques entre les sources dans le processus d'intégration de données. L'intégration de données est donc considérée comme automatique. Nous pouvons citer le projet PICSEL [64], OBSERVER [29], OntoBroker [65], KRAFT [32], COIN [6], etc. Comparées aux approches d'intégration utilisant des ontologies linguistiques, ces approches sont :

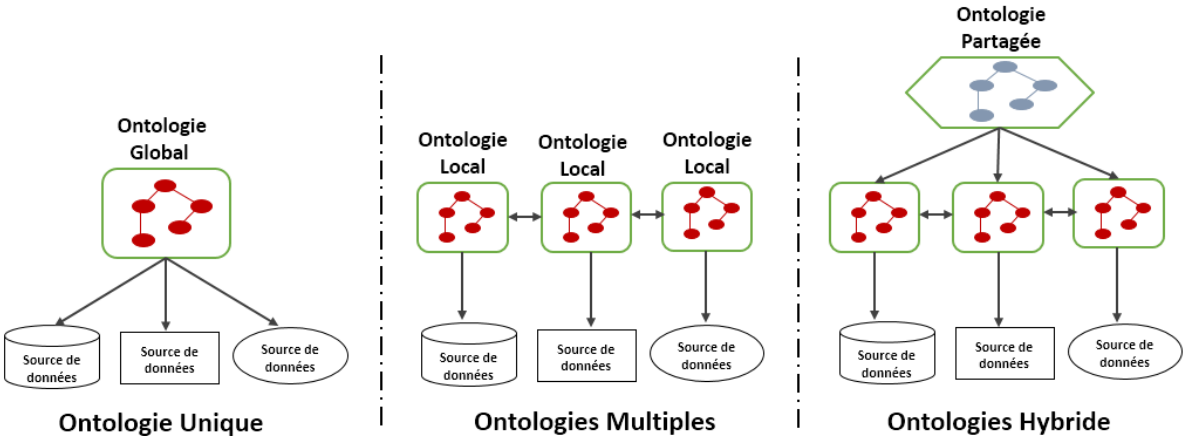


Fig. 1.10 – Différentes architectures d'intégration à base ontologique proposées par Wache et al. [70].

- Plus rigoureuses : car les concepts sont définis avec plus de précision, l'appariement peut être plus argumenté,
- Facilement outillées : si les différentes ontologies sont basées sur le même modèle, comme OWL [68], PLIB [69], etc. Une correspondance entre ontologies peut être exploitée par des programmes génériques pour intégrer les données correspondantes.

Plusieurs approches d'intégration à base ontologique ont été développées [70]. Ces dernières peuvent être divisées en trois catégories : approches avec une seule ontologie, approches avec ontologies multiples et approches hybrides (voir Figure 1.10). Dans l'approche avec une seule ontologie, chaque source référence la même ontologie globale de domaine. Cette approche est plus appropriée lorsqu'on veut intégrer des sources dans le même domaine d'application. Les systèmes d'intégration SIMS [30] et COIN [6] sont des exemples de systèmes utilisant cette approche. En conséquence, une nouvelle source ne peut ajouter aucun nouveau concept sans exiger le changement de l'ontologie globale. Dans l'approche à multiples ontologies (exemple du projet OBSERVER [29]), chaque source a sa propre ontologie développée indépendamment des autres sources. Dans ce cas, les correspondances inter-ontologies sont difficiles à mettre en œuvre. L'intégration des ontologies est donc faite d'une façon manuelle ou semi-automatique. [29]. Cette approche est plus appropriée lorsque on veut intégrer des sources qui appartiennent à différents domaines d'application.

Pour surmonter l'inconvénient des approches simples ou multiples d'ontologies, l'approche Hybride a été proposée. Dans cette dernière, chaque source a sa propre ontologie, mais toutes les ontologies utilisent un vocabulaire partagé commun (exemple du projet KRAFT [32]). Les approches d'intégration à base ontologique se divisent en deux catégories : (1) les approches sémantiques à posteriori, et les (2) approches sémantiques à priori.

Notre travail se positionne dans le domaine de l'intégration à base d'ontologies conceptuelles, nous discuterons le concept d'ontologie et ces approches de façon plus détaillée dans le prochain chapitre.

4.3.4 Discussion

Les aspects modélisation des données, c'est à dire comment intégrer les différents schémas des sources, et leur interrogation, et comment répondre efficacement aux requêtes posées sur le schéma global, ont été abondamment abordés dans la littérature [71].

Pour l'établissement des correspondances entre schémas, la plupart des systèmes sont basés sur une analyse manuelle du schéma effectuée par un expert, soit du domaine, soit d'intégration, même si des approches semi-automatiques de mise en correspondance sont parfois proposées, avec notamment l'utilisation de techniques issues de l'intelligence artificielle. Apporter une certaine automatisation dans le processus de prise en compte des sources permet une réduction et économie de temps de réponse et de coût.

4.4 Langages de représentation de données intégrées

Selon ce quatrième critère, les systèmes d'intégration sont structurés en fonction du langage de représentation des connaissances exprimant le schéma global. En effet, Un langage de représentation de données doit permettre de représenter la sémantique, la structure des données et des informations additionnelles (ontologies, requêtes).

4.4.1 Systèmes fondés sur un schéma global à base de règles

Les systèmes les plus représentatifs de cette famille sont : Razor [47], Internet Softbot [48], InfoMaster [27], Information Manifold [28]. Les systèmes Razor et Internet Softbot utilisant le langage DATALOG pour modéliser le schéma global et exprimer les requêtes de l'utilisateur. InfoMaster et Information Manifold utilisent des extensions de DATALOG. InfoMaster permet d'exprimer, en plus des règles, les contraintes d'intégrité. Le système HERMES [74] est un système de bases de données fédérées. Il ne repose pas sur la description sémantique d'un domaine d'application ou du contenu des différentes bases de données. Il repose sur un langage de requêtes uniforme permettant d'englober l'appel et de combiner dans une même requête l'appel à des bases de données de différents types.

4.4.2 Systèmes fondés sur un schéma global à base de classes

Le système TSIMMIS [26] est fondé sur un langage orienté objet appelé OEM, pour le schéma global et les vues, ainsi que sur le langage *OEM-QL* pour les requêtes. MOMIS [43] est fondé sur l'utilisation d'une logique de description très riche $ODLi^3$ pour décrire les schémas des sources de données à intégrer. SIMS [30] et OBSERVER [29] utilisent une logique de description pour décrire le schéma global, les vues et les requêtes, respectivement $\mathcal{LOOM}$ et $\mathcal{CLASSIC}$ [51].

4.4.3 Systèmes fondés sur un schéma global à base d'arbres

Le schéma global dans XYLEME [49] est un ensemble de DTDs qui sont des arbres de termes structurant le vocabulaire de domaine.

PICSEL [64] se distingue des systèmes existants par la possibilité d'exprimer le schéma global dans un langage $\mathcal{CARIN}$ combinant le pouvoir d'expression d'un formalisme à base de règles et un formalisme à base de classes. (Logique $\mathcal{ALN}$).

4.5 Niveaux d'abstraction

Selon ce cinquième critère, si plusieurs sources de données sont en cours d'intégration, cela peut être fait à un des nombreux niveaux. A chacun des niveaux, des méthodes d'intégration différentes peuvent être identifiées :

- *Au niveau des vues utilisateurs :* L'idée et le but derrière les méthodes qui sont utilisées pour intégrer les vues utilisateur est que les vues des différents utilisateurs du système doivent être intégrées dans un schéma de base de données commun.
- *Au niveau des schémas conceptuels :* les méthodes qui sont fondées sur des schémas conceptuels décrivent l'intégration des schémas de divers systèmes. Dans ce cas, le processus d'intégration doit faire face à l'hétérogénéité structurelle et sémantique.
- *Au niveau des données :* Le troisième type de méthodes opère généralement au niveau des données, ici, les méthodes sont basées sur des données concrètes de bases de données aux fins de l'intégration. Les méthodes d'intégration de données doivent faire face à deux principaux problèmes : l'identification d'entités identiques et la résolution des conflits entre les valeurs d'attribut.

4.6 Comparaison entre les différentes approches d'intégration

Dans le Tableau 1.2, on propose une classification des approches de mapping selon leur automaticité, scalabilité (passage à l'échelle), extension et maintenance et traitement des requêtes.

Tab. 1.2 – Comparaison entre les différentes approches d'intégration.

	Automaticité du Mapping			**Passage à l'échelle (Scalabilité)**	**Extension**	**Traitement**
	Sans Ontologie	**Ontologie Linguistique**	**Ontologie Formelle**			
GaV	Manuelle	Semi-automatique	Automatique	Difficile	M-a-j manuelle	Requêtes facile
LaV	Manuelle	Semi-automatique	Automatique	Oui	M-a-j manuelle	Réécriture difficile (vues)
GLaV	Manuelle	Semi-automatique	Automatique	Oui	M-a-j manuelle	Réécriture facile (vues)
BGLaV	Manuelle	Semi-automatique	Automatique	Oui	M-a-j manuelle	Réécriture facile (vues)
BaV	Manuelle	Semi-automatique	Automatique	Oui	M-a-j manuelle	A prouver

5. Traitement des requêtes dans un système d'intégration

L'exécution des requêtes dans des bases de données hétérogènes et distribuées conduit à de nombreux problèmes qui peuvent se classer en deux catégories : l'intégration des données hétérogènes et le traitement des requêtes. L'intégration des données hétérogènes est confrontée à de nombreux conflits de données (conflits syntaxiques, schématiques et sémantiques). Le traitement d'une requête posée indépendamment de la localisation des différentes sources de données induit aux difficultés suivantes : (1) la reformulation d'une requête, (2) la décomposition d'une requête et (3) la recomposition des résultats et le niveau de l'optimisation.

Le rôle d'un système de médiation consiste à reformuler la requête d'un utilisateur en termes de schémas sources, puis à la décomposer en sous-requêtes, qui seront envoyées aux différentes sources. Les résultats sont ensuite renvoyés au médiateur, qui les intègre avant de les renvoyer à l'utilisateur.

Dans un système d'intégration, l'interrogation s'effectue généralement en utilisant des requêtes conjonctives à base de règles (des requêtes de type sélection-projection-jointure) [72] définies à l'aide du vocabulaire du schéma global qui exprime les vues sur les différentes sources. On parle alors d'interrogation basée sur les vues [73].

Étant donné un schéma *S*, une requête conjonctive est de la forme $rep(x) \leftarrow R_1(x_1), ...,R_n(x_n)$ Où :

- $rep(x)$ est appelée la tête de la règle,
- $R_1(x_1), ...,R_n(x_n)$ est appelé le corps de la règle,
- Les $R(x_i)$ sont appelés des atomes,
- R_i est un nom de prédicat de *S*,
- Chaque x_i est une expression de la forme $e_1, ..., em_i$,
- Chaque e_j est soit une variable, soit une constante de domaine.

Dans un système d'intégration, les requêtes étant posées sur le schéma médiateur ou schéma virtuel en utilisant un certain nombre de vues, le système essaie d'effectuer la réécriture des requêtes des utilisateurs en s'assurant que les requêtes réécrites sont soit équivalentes aux requêtes initiales, soit incluses (de façon maximale) dans chacune des requêtes initiales. La réécriture maximale de requête essaie de trouver la meilleure solution possible par rapport aux sources disponibles.

Étant donné un schéma *S* et son extension, une requête *A* sur *S* inclut une requête *B* sur *S*, dénoté $A \subset B$, si le résultat de l'exécution de la requête *B* est un sous-ensemble du résultat de l'exécution de la requête *A*, et ce pour n'importe quelle extension de *S*. Les deux requêtes *A et B* sont dites équivalentes si les deux ensembles résultats sont égaux quelle que soit l'extension du schéma.

L'algorithme jugé le plus performant pour la réécriture de requêtes est l'algorithme MiniCon [71]. Cet algorithme résulte de la combinaison de deux algorithmes. Le premier nommé *bucket* a été développé dans le cadre du système Information Manifold [28]. Le second, nommé *inverse-rules*, a été mis en œuvre dans le système InfoMaster [27]. Les requêtes et les vues acceptées par MiniCon sont des requêtes conjonctives avec des prédicats de comparaison arithmétique.

6. Principaux systèmes d'intégration

Dans cette section, nous décrivons sommairement quelques systèmes d'intégration. Nous ne pouvons tous les présenter, mais les systèmes choisis sont représentatifs de leur famille puis nous adressons un tableau qui récapitule leurs caractéristiques.

6.1 Systèmes d'intégration

Nous allons discuter rapidement les systèmes suivants : TISMMIS [26] qui est considéré comme un des premiers projets d'intégration de données, PIAZZA [42] comme exemple de système utilisant plusieurs médiateurs. SIMS [30] qui utilise une ontologie globale et l'approche centrée requête (GaV). MOMIS [43] comme exemple de système utilisant une approche de découverte semi-automatique de correspondances. OBSERVER [29] comme exemple de système utilisant plusieurs ontologies. PICSEL [64] qui repose sur la combinaison d'un formalisme à base de règles et d'un formalisme à base de classes, et enfin XYLEME [80] comme exemple de système Hybride d'intégration de données semi-structurées.

- **TSIMMIS** [26] (The Stanford-IBM Manager of Multiple Information Sources) est un système destiné à l'intégration (manuelle) de plusieurs sources hétérogènes. C'est un système fédéré basé sur l'architecture médiateur/adaptateurs utilisant l'approche centrée requête. Le modèle de données utilisé pour décrire le contenu des sources et pour exprimer les requêtes est l'Object Exchange Model. Chaque adaptateur est un programme qui s'occupe d'une source particulière. Il reçoit les requêtes posées dans le langage OEM et les transforme en langage compréhensible par la source. Les sources traitées sont structurées et semi-structurées. TSIMMIS ne fournit pas de schéma global unique et utilise plusieurs médiateurs.
- **SIMS** [30] (Single Interface to Multiple Sources) est un système qui vise l'intégration de sources de données hétérogènes et de bases de connaissances qu'il représente en utilisant le langage *LOOM*, basé sur la logique de description. SIMS adopte l'architecture à ontologie unique et utilise le Mapping GaV. Le médiateur dans SIMS est spécialisé dans un seul domaine d'application.
- **MOMIS** [43] (Mediator envirOnment for Multiple Information Sources) est un système d'intégration qui se base sur son propre langage orienté objet dénommé *ODLi*^{3}. Il utilise une ontologie unique globale appelée GVV (Global Virtual View) qui est générée semi-automatique. MOMIS adopte l'approche GaV pour le mapping entre l'ontologie globale et les sources locales.
- **OBSERVER** [29] (Ontology Based System Enhanced with Relationships for Vocabulary hEterogeneity Resolution) est un système qui permet l'interopérabilité entre différentes sources, en utilisant pour cela de multiples ontologies pour décrire les sources de données. Il se base sur *CLASSIC* [51], un langage de logique de description. Il n'y a pas d'ontologie

globale dans OBSERVER ; le mapping entre les multiples ontologies est réalisé à l'aide de tables de correspondance. Cependant, les relations entre ontologies sont limitées à des relations lexicales basiques telles que les synonymes, hyponymes et hyperonymes.

- **PICSEL** [64] (Production d'Interface à base de Connaissance pour des Services En Ligne) est un système qui a connu plusieurs versions. Il vise à construire un médiateur à base de connaissances. Le langage utilisé est *ALN-CARIN*, un formalisme à base de règles et de logique de description. Dans sa dernière version, PICSEL adopte l'approche Hybride comme architecture d'ontologie et utilise le mapping LaV entre l'ontologie globale et les ontologies locales. Ce mapping est généré semi automatiquement.
- **XYLEME** [80] (For XML Warehouse Integration) est un système d'intégration physique Hybride avec XML comme modèle de données. Il adopte l'approche Hybride dans son architecture. Les ontologies globale et locales sont exprimées à l'aide d'arbres, avec un mapping GaV / LaV. Le mapping est réalisé semi-automatiquement par la génération de tables de correspondance entre les chemins de l'ontologie globale et les chemins des ontologies locales.
- **PIAZZA** [42] (The Piazza Peer Data Management Project) en dernier, est un système qui intègre des sources qui sont, soit des ontologies, soit des données semi structurées (décrites par schéma XML ou DTD). Il s'appuie sur le modèle de données XML et une adaptation de *XQuery* comme langage de requêtes. Ce système suit une architecture P2P (Peer-to-Peer) avec des ontologies et/ou schémas multiples. Le mapping entre les différentes ontologies est bidirectionnel (une source est décrite en fonction d'une autre et vice versa). Pour une requête donnée, le schéma d'un nœud (source) peut être considéré comme étant le schéma global et ainsi, de proche en proche, les sources concernées seront interrogées.

6.2 Comparaison des principaux systèmes d'intégration

Dans le Tableau 1.3 on compare les principaux systèmes d'intégration selon les critères de classification : (1) Localisation de données (2) Sens de mise en correspondance (3) Automaticité d'intégration (4) Architecture Ontologique (5) et Représentation des données d'intégration.

La plupart de ces systèmes utilisent une architecture à ontologie unique. Cette approche est intéressante dans le cas où les sources auraient une vue similaire du domaine d'application. Si ce n'est pas le cas, il est difficile de trouver une ontologie consensuelle minimale. D'un autre côté, une source ne peut être changée sans reconsidérer l'ontologie globale et les autres sources pour s'assurer qu'il n'y a pas de conflits. Ceci ne privilégie pas l'autonomie des sources. Pour les autres systèmes qui sont basés sur des ontologies multiples, la comparaison d'ontologies est difficile en l'absence d'un vocabulaire commun. De plus, ils ont besoin de définir le mapping entre plusieurs couples d'ontologies. L'approche GaV est la seule technique facilitant la reformulation des requêtes. De plus, dans GaV, une automaticité de mapping pourrait être assurée en utilisant des ontologies formelles. Reste à prouver la possibilité du passage à l'échelle.

XYLEME [49] et PICSEL [64] sont les seuls systèmes à utiliser l'approche Hybride. Mais le premier adopte l'approche entrepôt qui ne favorise pas la fraîcheur des données. Quant au second, il utilise son propre formalisme de langage de description. L'intégration de sources de données à base ontologique par l'approche matérialisée (entrepôt de données) a constitué le travail de thèse de Dung Nguyen [33].

Tab. 1.3 – Comparaison des principaux systèmes d'intégration.

Système d'intégration	Architecture d'intégration	Nature du mapping	Automaticité d'intégration	Architecture Ontologique	Représentation données
TSIMMIS	Virtuelle	GaV	Manuel	Sans ontologie	Base de classes
OBSERVER	Virtuelle	GaV	Semi-automatique	Ontologies multiples	Base de classes
MOMIS	Virtuelle	GaV	Semi-automatique	Ontologie unique	Base de classes
SIMS	Virtuelle	GaV	Automatique	Ontologie unique	Base de classes
COIN	Virtuelle	GaV	Automatique	Ontologie unique	Base de classes
KRAFT	Virtuelle	LaV	Semi-automatique	Ontologie Hybrides	Base de classes
PICSEL	Virtuelle	LaV	Semi-automatique	Ontologie unique	Base de règles
ManiFold	Virtuelle	LaV	Semi-automatique	Ontologies multiples	Base de règles
InfoMaster	Virtuelle	LaV	Semi-automatique	Sans ontologie	Base de règles
PLIB	Entrepôt	GaV	Automatique	Ontologie Hybrides	Base de classes
OntoDaWa	Entrepôt	GaV	Automatique	Ontologie Hybrides	Base de classes
PICSEL2	Hybride	LaV	Automatique	Ontologie unique	Base de règles
XYLEME	Hybride	GaV / LaV	Semi-automatique	Ontologie Hybrides	Base d'arbres
PIAZZA	Peer-to-Peer	LaV	Semi-automatique	Ontologies multiples	Base de règles

7. Positionnement de notre approche

Notre domaine d'application cible est celui de l'éducation supérieure et les universités. Il s'agit d'intégrer automatiquement les sources de données des différentes universités à travers une vue unifiée, en offrant, sans aucune programmation, des possibilités d'accès ergonomiques.

L'état de l'art que nous avons présenté montre que nous pouvons atteindre une intégration automatique de différentes sources de données. Il s'agit d'adopter une architecture médiateur et utiliser des ontologies formelles (conceptuelles) dans une perspective d'intégration sémantique. Nous avons utilisé une ontologie de domaine unique. Ce choix est motivé par le fait que les sources de données appartiennent au même univers et ont une vue similaire du domaine d'application. Ce travail s'inscrit dans le contexte d'une approche d'intégration à base ontologique se basant sur les principes suivants :

1. Il existe une ontologie de domaine mise au point par des experts du domaine et se compose de tous les termes et les concepts existants dans un domaine spécifique, et
2. Chaque base de données contient outre ses données, son schéma de métadonnées qui en définit ses attributs et les relations afin de mener à bien le processus de mapping.

Notre système d'intégration consiste à permettre à la fois : (1) à chaque concepteur (Université) de choisir librement son schéma, et (2) réaliser l'intégration automatique des différentes bases de données.

Dans notre travail, l'approche LaV est adoptée. Ce choix est essentiel pour assurer le passage à l'échelle (Scalabilité) dans un domaine où les sources de données sont en nombre important. (voir Figure1.11).

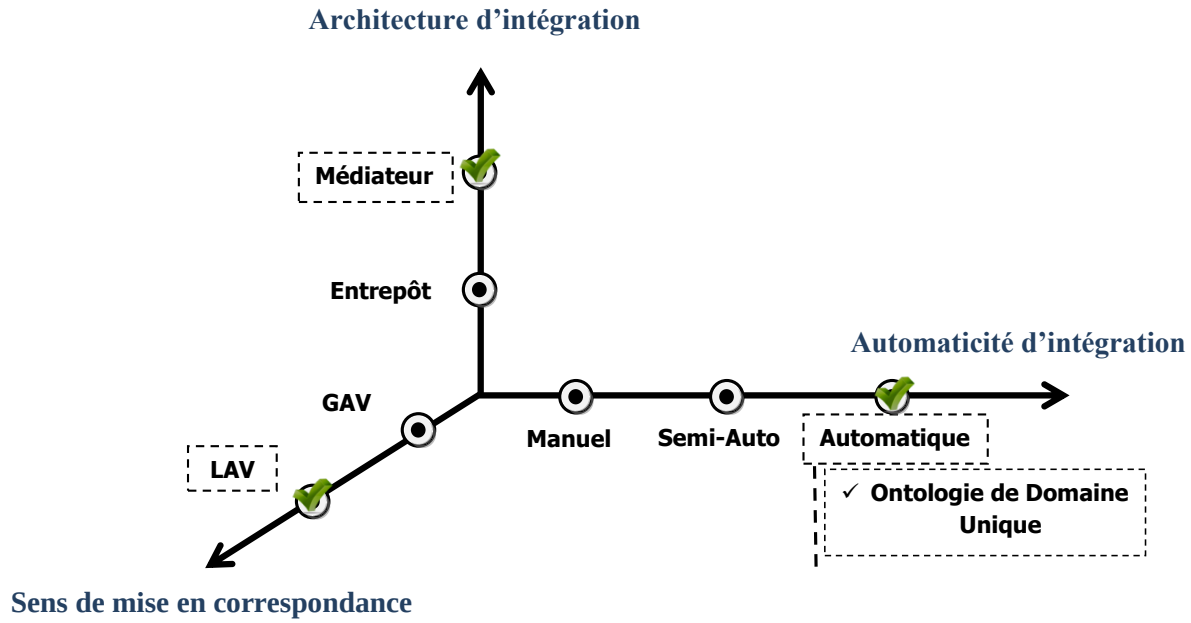


Fig. 1.11 – Positionnement de notre approche.

8. Conclusion

Dans ce chapitre, nous avons défini la problématique de l'intégration de données qui consiste principalement à préserver l'autonomie et traiter l'hétérogénéité notamment sémantique des sources de données. Cette hétérogénéité provient des choix différents qui sont faits pour représenter des faits du monde réel dans un format informatique. La question fondamentale lorsque l'on veut faire inter-opérer des bases de données hétérogènes en identifiant les conflits entre les concepts dans des sources différentes qui peuvent avoir des liens sémantiques entre eux. Une taxonomie des conflits sémantiques qu'il convient de résoudre a été présentée : (1) conflits de représentations, (2) conflits de noms, (3) conflits de contextes, (4) conflits de mesure de valeur.

Nous avons défini la notion d'un système d'intégration, ses différents composants et ses tâches principales. On a résumé les différentes phases du processus d'intégration des bases de données hétérogènes qui sont. (1) Pré-intégration qui commence par définir les langages et outils de modélisation. (2) Comparaison qui compare les schémas des bases de données existantes, en cherchant des portions de schémas qui soient intéressantes pour l'application cible. (3) Mise en conformité des schémas qui traite les conflits induits par les correspondances, en suivant les objectifs fixés dans la phase de Pré-intégration. (4) Fusion et Restructuration qui produisent les schémas des solutions intégrées, et indiquent comment accéder aux données intégrées et les manipuler.

Nous avons présenté une classification des approches d'intégration de données existantes. Cette classification est faite en se basant sur les cinq critères orthogonaux suivants : (1) la représentation des données intégrées, (2) le sens de la mise en correspondance entre schéma global et schémas locaux, (3) la nature du processus d'intégration, (4) les langages de représentation des données intégrées, et (5) le niveau d'abstraction. Le premier critère nous permet de déterminer le type de stockage des données du système d'intégration qui est virtuel ou matériel. Le deuxième nous permet d'identifier le sens de mise en correspondance entre schéma global et schémas locaux. Ce sens influence directement la possibilité de passage à l'échelle et la complexité de traitement de requête du système d'intégration. Le troisième critère spécifie si le processus d'intégration de données est effectué d'une façon manuelle, semi-automatique ou automatique. Ce critère devient essentiel lorsque l'on veut intégrer un nombre important de sources de données indépendantes. Le quatrième critère influence le processus d'intégration car il spécifie la syntaxe à partir de laquelle les sources et les requêtes vont être décrites. Le dernier critère spécifie les niveaux d'abstraction, si plusieurs systèmes sont en cours d'intégration, cela peut être fait à un des nombreux niveaux. Nous avons comparé entre les différentes approches d'intégration selon leur automaticité, scalabilité, traitement des requêtes extension et maintenance.

Nous avons montré la façon dont les systèmes d'intégration traitent les requêtes. Nous avons présenté et classifier les principaux systèmes d'intégration et enfin nous avons positionné notre approche. Dans le chapitre suivant nous présentons le concept d'ontologie et son exploitation dans le cadre de l'intégration des données.

Chapitre 2
Technologie des ontologies et intégration des données

« C'est sur son utilité que la vérité fonde sa valeur et ses droits ; elle peut être quelquefois désagréable à quelques individus et contraire à leurs intérêts, mais elle sera toujours utile à l'espèce humaine ...

L'utilité est donc la pierre de touche des systèmes, des opinions et des actions des hommes.»

Paul–Henri

Chapitre II

Technologie des ontologies et intégration des données

Sommaire

1 Introduction 41
2 La notion d'ontologie 42
2.1 Définition 42
2.2 Ontologies en Informatique 42
2.2.1 Définition 42
2.2.2 Concepts primitifs et concepts définis 43
3 Exemples de formalismes d'ontologies 44
3.1 Formalismes d'ontologies orientés gestion et échange de données 44
3.1.1 RDF/ RDF-Schéma 45
3.1.2 PLIB 49
3.2 Formalismes d'ontologies orientés inférence 52
3.2.1 DAML-ONT, OIL, DAML+OIL 53
3.2.2 OWL 53
4 Similitudes et différences des formalismes d'ontologie 57
4.1 Similitudes 57
4.2 Différences 57
4.3 Le modèle en oignon 58
4.3.1 Ontologies Conceptuelles Canoniques 59
4.3.2 Ontologies Conceptuelles Non-canoniques 60
4.3.3 Ontologies Linguistiques 61
5 Ontologie conceptuelle et Intégration des données 62
5.1 Différence entre ontologie/modèle conceptuel 62
5.2 Utilisation d'ontologies conceptuelles pour l'intégration de données 64
5.2.1 Intégration sémantique à posteriori 65
5.2.2 Intégration sémantique à priori 68
6 Conclusion 71

1. Introduction

Parallèlement à l'explosion de la quantité d'information numérique dans de nombreux domaines au cours des dernières années, de nombreux travaux ont été menés pour développer des méthodes permettant de représenter explicitement la signification de ces données sous des formes échangeables et exploitables par des ordinateurs.

Les ontologies sont des structures qui permettent de représenter explicitement la sémantique d'un domaine par des modèles objets consensuels dont chaque concept (classe ou propriété) est associé un identificateur universel permettant de référencer la sémantique qui lui correspond.

Les ontologies, définies par Gruber [81] comme une spécification explicite d'une conceptualisation, se sont imposées comme un moyen pour expliciter la sémantique des données. Elles permettent aux programmes d'échanger cette sémantique et, le cas échéant, de réaliser des raisonnements et des traitements intelligents sur les données.

Les ontologies sont aujourd'hui utilisées dans un nombre croissant d'applications, par exemple pour faciliter la recherche d'information dans le domaine du Web en annotant les documents par des instances ontologiques, ou, dans le domaine technique, pour représenter des catalogues de composants industriels. Par instance ontologique, nous entendons un objet dont le sens est défini par son appartenance à une classe ontologique et par les valeurs d'un certain nombre de propriétés définies dans la même ontologie. Nous appellerons données à base ontologiques (DBO) un ensemble d'instances ontologiques.

Dans ce chapitre, nous définissons la notion d'ontologie et les problèmes liés à son exploitation dans les bases de données. Nous présentons deux catégories de formalisme d'expression des ontologies et les problèmes qu'elles visent à résoudre au travers des constructeurs qu'elles offrent. Ces deux formalismes sont PLIB, d'une part, adapté au domaine technique, et OWL d'autre part développé pour faciliter l'accès aux données du Web. Ces formalismes sont les plus utilisés. Nous présentons ensuite, une comparaison de ces formalismes. Cette comparaison permet de montrer la complémentarité des différents formalismes et amener à une classification des ontologies. Nous achevons le chapitre avec l’apport des ontologies conceptuelles dans l’élaboration des systèmes d’intégration de données et une conclusion.

2. La notion d'ontologie

La notion d'ontologie est apparue en informatique dans les années quatre-vingt-dix. Une ontologie est essentiellement une représentation explicite de la conceptualisation d'un domaine, telle qu'elle est perçue par une communauté donnée [81]. Les ontologies sont aujourd'hui utilisées dans divers domaines et on retrouve dans la littérature [5] de nombreuses définitions du terme ontologie en fonction du secteur d'activité visé [82] (intégration de données, indexation de document, traitement automatique de la langue naturelle (TALN), etc.). En ce qui nous concerne, nous adoptons la définition suivante :

2.1 Définition

> *Une ontologie est une représentation formelle, explicite, référençable et consensuelle de l'ensemble des concepts partagés d'un domaine en termes de classes d'appartenance et de propriétés caractéristiques* [83].

Cette définition met en avant trois caractéristiques qui distinguent une ontologie de domaine des autres modèles informatiques tels que les modèles conceptuels et les modèles de connaissance. Une ontologie est une représentation :

- *formelle*, exprimée dans un langage de syntaxe et de sémantique formalisé (RDF schéma [84], DAML+OIL [85], OWL [68], PLIB [87], etc.) permettant ainsi des raisonnements automatiques ayant pour objet soit d'effectuer des vérifications de consistance, soit d'inférer de nouveaux faits ;
- *consensuelle*, c'est-à-dire admise par l'ensemble des membres (et des systèmes) d'une communauté et;
- *référençable*, c'est-à-dire que toute entité ou relation décrite dans l'ontologie peut être directement référencée par un symbole « identifiant », à partir de n'importe quel contexte, afin d'expliciter la sémantique de l'élément référençant.

2.2 Ontologies en Informatique

2.2.1 Définition

D'un point de vue pratique, une ontologie informatique est formalisée en utilisant cinq concepts :

1. Les **classes** (catégories d'objets modélisés qui ont une existence propre dans le domaine modélisé). Une classe est une collection d'objets définie en intention. Exemple : Territoire, Document, Personne, etc.
2. Les **propriétés** (attributs des objets modélisés qui permettent de caractériser une instance d'une classe). Une propriété est une relation binaire entre deux classes ou entre une classe et un domaine de valeurs.
3. Les **domaines de valeurs** (ou types de données) : un domaine de valeurs est un ensemble mathématique défini en extension ou en intention. Exemple : réels, caractères, booléen, mais aussi les types structurés : liste, tableau, ….etc.
4. Les **individus (ou instances)** : un individu du domaine modélisé est représenté de façon ontologique comme une instance appartenant à une classe définie dans l'ontologie. Par exemple, l'individu *Maissa Atman* peut être défini ontologiquement comme une instance de la classe des

[5] http://websemnantique.org/Ontologie

Personnes, avec la valeur ‘2010’ pour la propriété « année de naissance » et, la valeur ’féminin’ pour la propriété « genre ». La classe des Personnes et les propriétés « année de naissance » et « genre » sont définies dans l'ontologie.

5. Les **axiomes** qui sont des prédicats s'appliquant sur les relations, les classes ou les instances et permettent d'assurer l'intégrité des données ou de faire des inférences. Exemple : l'âge d’une personne doit être une valeur positive, une instance ne peut être caractérisée que par les propriétés applicables à sa classe.

2.2.2 Concepts primitifs et concepts définis

La conception d'une ontologie se fait en exploitant un formalisme d'expression d'ontologies. Ces formalismes utilisent des représentations orientées objet. Ils permettent de définir des constructeurs en vue de créer des ontologies au travers des concepts énumérés ci-dessus. Depuis leur émergence en informatique, plusieurs courants ont été suivis pour définir les ontologies. On peut distinguer deux principaux courants :

1. Les formalismes d'ontologies orientés gestion et échange de données qui visent à représenter la sémantique des données d'un domaine d'application de manière précise et unique de façon à permettre le partage et l'échange d'information.
2. Les formalismes d'ontologies orientés inférence qui visent à permettre certains raisonnements sur un domaine d'application pour résoudre certains problèmes, en exploitant des connaissances relatives au domaine étudié.

Les formalismes d'ontologie orientés gestion et échange de données définissent des constructeurs qui vont permettre de décrire les concepts des ontologies de manière à faciliter l'échange d'information. De ce fait, ces ontologies permettent de représenter chaque information de manière unique ou canonique. Nous caractérisons les ontologies définies par ces formalismes des ontologies canoniques.

Au contraire, les formalismes d'ontologie orientés inférence définissent plusieurs constructeurs qui vont permettent de définir le même concept (classes, propriétés et instances) de l'ontologie. Nous appelons ces constructeurs *les constructeurs d'équivalence conceptuelle* car ils offrent le moyen de représenter de différentes manières le même concept ou la même information. Par exemple, l’individu *Maissa Atman* peut être décrit comme une instance de la classe *Personne* avec les caractéristiques : *nom* et *genre*, ou encore comme une instance de la classe *Femme* si cette dernière est également définie au niveau de l'ontologie comme équivalente à l'ensemble des personnes pour lesquelles la valeur de la propriété *genre* est *féminin*. Ainsi, *Maissa Atman* peut être décrite par deux représentations équivalentes qui permettent de faire automatiquement des inférences entre elles. Les ontologies possédant de telles constructions sont dites non-canoniques.

Deux catégories de concepts sont offertes par les différents formalismes d’ontologies :

1. *les concepts primitifs ou canoniques* qui sont des concepts pour lesquels nous ne sommes pas capables de donner une définition axiomatique complète [81]. La définition de ces concepts s'appuie sur une documentation textuelle et un savoir partagé avec les utilisateurs mais ne peut se réduire avec d'autres concepts. L'ensemble des concepts primitifs suffit pour définir les frontières du

domaine conceptualisé par une ontologie. Les concepts primitifs sont la fondation sur laquelle d'autres concepts de l'ontologie pourront ensuite être définis par équivalence, si besoin est. La définition de concepts primitifs étant toujours, au moins partiellement, informelle, le seul critère de qualité pour une telle définition, est qu'elle représente un consensus parmi une communauté.

2. *les concepts définis* ou *non- canoniques* qui sont les concepts pour lesquels une ontologie fournit une définition axiomatique complète au moyen de conditions nécessaires et suffisantes exprimées en termes d'autres concepts primitifs ou eux-mêmes définis [81]. Les définitions de tels concepts sont conservatives car elles associent un nouveau concept à quelque chose qui est déjà défini par un autre moyen dans l'ontologie en cours de conception. Elles n'introduisent donc pas de nouvelles instances mais des alternatives de désignation pour des concepts que l'on pouvait déjà désigner. Elles n'enrichissent donc pas les connaissances sur ce domaine, mais le vocabulaire qui permet de les représenter. Dans ce type de formalisme d'ontologie ou les langages de modélisation permettent non seulement de décrire des concepts primitifs, mais comportent également différents opérateurs permettant de composer les concepts pour construire des concepts définis, cette caractéristique est la base des mécanismes d'inférence. Par exemple, un système d'inférence permettra de décider qu'une *Femme* est une *Personne* et qu'une *Personne* de *genre féminin* est une *Femme*. Il permettra également d'effectuer des classifications automatiques, c'est à dire de déduire automatiquement des relations de subsomption entre concepts, des relations d'équivalence conceptuelle etc. Il permettra également de calculer l'appartenance d'instances à certaines classes, à partir de la définition axiomatique complète des concepts définis.

Nous présentons dans les sections suivantes quelques formalismes qui permettent de construire les ontologies.

3. Exemples de formalismes d'ontologies

Les ontologies sont exprimées à partir de formalismes d'ontologies. Ces formalismes offrent des constructeurs permettant de définir les différents concepts (classes, propriétés, types de données, individus, ...etc.) que l'on retrouve dans une ontologie. Les ontologies sont utilisées dans différentes disciplines (intégration des bases de données, TALN, recherche d'information, ...) et ce, avec des objectifs différents. Il existe donc différents formalismes de définition d'ontologies suivant les objectifs spécifiques visés par les différentes spécialités. Nous introduisons dans cette section, les principaux formalismes d'ontologies. L'objectif est de ressortir (1) ce qu'ils ont en commun et (2) ce qui les distinguent. Cette illustration nous permet d'identifier le but, les points forts et les points faibles de chacun.

3.1 Formalismes d'ontologies orientés gestion et échange de données

Dans cette section, nous présentons les formalismes d'ontologie RDF-Schéma et PLIB. Ces formalismes offrent respectivement dans le domaine du Web et de l'ingénierie, des constructeurs permettant de représenter les informations de l'univers du discours de manière à faciliter le partage et l'échange des ontologies et des instances associées.

3.1.1 RDF/ RDF-Schéma

Il s'agit ici de deux formalismes développés pour expliciter par annotation (une partie) de la sémantique du Web. RDF définit à la fois une syntaxe, utilisable ensuite pour tous les formalismes du Web (dont RDF-Schéma et OWL) et un mécanisme d'annotation permettant ensuite d'annoter les éléments existants du Web, qualifiés de ressources. Il ne s'agit donc pas d'un formalisme d'ontologie.

RDF-Schéma étend RDF pour permettre la définition des classes et des propriétés devenant donc un formalisme simple de définition d'ontologies [17]. RDF-Schéma peut utiliser la syntaxe RDF. Il utilise aussi RDF pour annoter les ressources du Web qui seront représentées comme instances des classes de l'ontologie RDF-Schéma.

- **RDF**

RDF (Resource Description Framework) est le premier langage apparu pour définir la sémantique sur le Web. A l'origine, il était essentiellement destiné à associer aux documents du Web des annotations sémantiques (titre, auteur, ...) exploitables par machine. Puis, l'utilisation des annotations a été étendue à toute information pouvant être référencée sur le Web (Site Web complet, page Web, ou encore un élément particulier d'une page Web) et, l'information que l'on voulait représenter a été étendue à la signification de ressource Web. La syntaxe du formalisme RDF est utilisée par les autres formalismes d'ontologies Web en particulier pour décrire les individus des classes des ontologies.

Le développement de RDF a été motivé par la perspective de :

- manipuler des métadonnées Web, afin de fournir des informations sur les ressources Web et les systèmes qui les utilisent ;
- faciliter la recherche et le traitement automatique de l'information du Web par la coopération (indexation, classement, diffusion, ...) des agents logiciels qui exploitent ces métadonnées.

Un modèle RDF est défini à partir de quatre ensembles :

1. Les *Ressources* : Une ressource est tout élément que l'on peut référencer par un identifiant appelé Uniform Resource Identifier (URI).
2. Les *Littéraux* : qui sont des valeurs éventuellement typées par un des types de données primitif défini par XML schéma.
3. Les *prédicats* : Un prédicat est une propriété, un aspect, une caractéristique, un attribut ou une relation spécifique que l'on peut utiliser pour décrire une ressource.
4. Les *déclarations* : qui permettent de décrire tout élément selon un mécanisme particulièrement simple. Une déclaration est un triplet de la forme : *sujet, prédicat, objet ;* où *sujet* est une ressource, *prédicat* est une propriété, et *objet* est soit une ressource, soit un *littéral*. Par exemple : *L'Algérie est un Territoire* est une déclaration RDF possible.

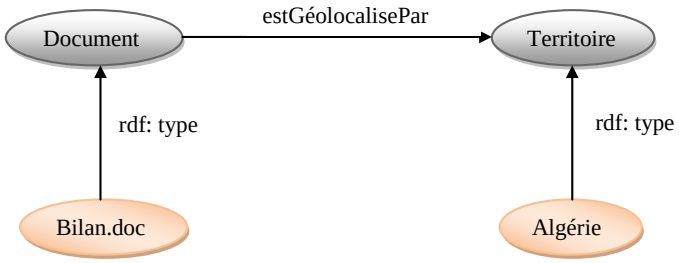


Fig. 2.1 – Exemple d'un graphe RDF.

Exemple : La Figure 2.1 illustre un exemple de graphe RDF. Les deux nœuds *Territoire* et *Document* représentent des ressources, et le nœud *Algérie* est un littéral. Les arcs représentent les prédicats. Chaque arc est orienté du sujet vers l'objet de la déclaration.

RDF représente toute information par un ensemble de déclarations. Il ne permet cependant pas de définir des vocabulaires permettant de formuler des contraintes sémantiques plus riches (par exemple, de définir l'ensemble des valeurs permises pour une propriété) ou de faire des raisonnements. De plus, RDF ne permet pas de catégoriser le domaine modélisé en termes de classes et de propriétés. Ce n'est donc pas un formalisme d'ontologie. Il fournit par contre une syntaxe et un langage simple pour annoter les ressources du Web. Ce langage est notamment utilisé :

- pour représenter les constructions de l'ensemble des langages d'annotation du Web (XML, HTML).
- par les langages d'ontologies du Web (DAML+OIL, RDF-Schéma, OWL) pour représenter les caractéristiques des ressources en représentant celles-ci comme des instances de classe d'ontologie.

Afin de caractériser l'ensemble de ressources du Web, le formalisme RDFS-Schéma a été proposé.

- **RDF-Schéma**

RDF-Schéma, aussi appelé RDFS est le premier formalisme d'ontologie défini sur le Web. RDF-Schéma fournit les prédicats essentiels pour représenter (en RDF) une ontologie. Ces prédicats prédéfinis pourront alors être utilisés afin de définir des ontologies RDFS (aussi appelé vocabulaire RDF) et caractériser ainsi les ressources du Web. RDF-Schéma est un des piliers du Web sémantique. Grâce à RDF-Schéma, il est par exemple possible de définir que le concept de *Territoire* dans le vocabulaire intitulé «espace géographique», représente une zone géographique. Une fois que ce vocabulaire est formellement défini grâce à RDF-Schéma, n'importe quel outil peut désormais utiliser le fait que *Territoire* est un cas particulier de zone géographique. Les données de tels outils pourront être publiées sur Internet et faire l'objet d'une indexation par un autre outil connaissant ce vocabulaire : les utilisateurs de ce dernier outil pourront donc par exemple, lister tous les documents qui font référence à (sont géo-localisés par) la zone géographique Algérie. RDF-Schéma est un système de typage pour RDF. L'utilisation conjointe de RDF et RDF-Schéma dans le Web Sémantique permet donc à la fois de représenter (en RDF-Schéma) une ontologie et (en RDF) des instances définies en termes de cette ontologie.

RDF-Schéma est le plus simple formalisme d'ontologie. Il est doté du nombre minimum de constructeurs nécessaires à la définition d'une ontologie. Ces constructeurs vont être retrouvés dans tous les autres langages d'ontologie du Web (DAML+OIL, OWL).

Constructeurs de classes

Pour modéliser les ressources du Web, RDFS permet de :

- définir des classes *(rdfs : class) :* une classe est un ensemble défini en intension de plusieurs objets analogues d'un certain point de vue et que l'on souhaite regrouper. Exemple : la classe des territoires.
- organiser les classes en une hiérarchie de spécialisation *(rdfs : subclassOf).* Exemple la classe *Commune* peut être définie comme une sous-classe de la classe *Territoire*.

Constructeurs de propriétés

RDFS permet aussi de :

- définir des propriétés *(rdf : property).* Exemple : *code_ ISO* est une propriété de la classe *Territoire.*
- organiser les propriétés en une hiérarchie de spécialisation *((rdfs : subProperty)).* Exemple : *a_ pour_ fils* est une sous-propriété de *a_ pour_ enfants.*

RDFS définit également :

- La ou les classes auxquelles on peut affecter une propriété et, qui constituent le domaine[6] de la propriété *(rdfs : domain)* (par exemple : la classe *Document* peut être l'objet de la propriété *estGeolocalisePar*). Lorsque le domaine d'une propriété n'est pas défini, cette dernière peut être utilisée pour décrire n'importe quelle instance appartenant à l'univers du discours conceptualisé par l'ontologie.
- le Co-domaine, ou le domaine de valeurs d'une propriété, *(rdfs : range)* (par exemple : la propriété *estGeolocalisePar* à pour valeur un élément de la classe *Territoire*). Comme la définition du domaine, la définition du Co-domaine d'une propriété peut être définie par une ou plusieurs classes (avec le sens d'intersection) [7], les types de données peuvent également être utilisés.

Types de données

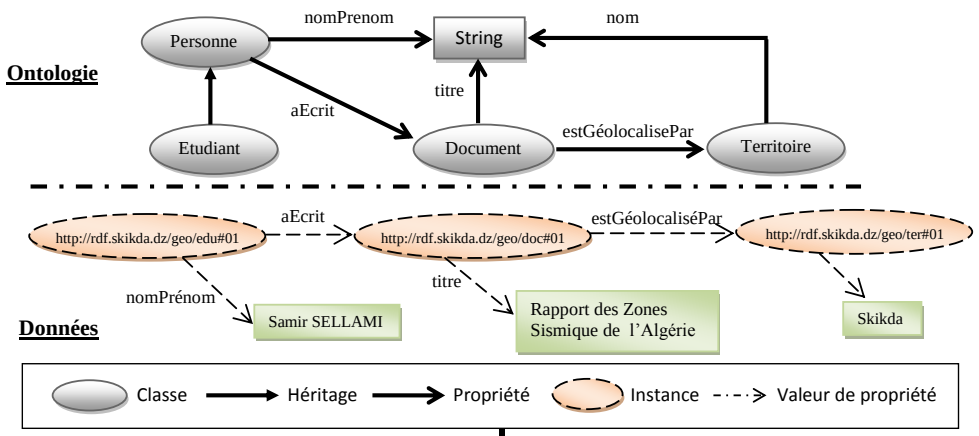
Les valeurs d'une propriété peuvent être des instances de classes ou des littéraux. Ces littéraux peuvent être typés en utilisant les types de données prédéfinis de XML schéma. Ceci permet par exemple de représenter des valeurs de types chaîne de caractères, numérique ou date. Par ailleurs, RDFS fournit également des types collections *(rdfs : Container).* RDFS utilise des constructeurs de collections définis dans RDF, en particulier les listes *(rdf : List)* et les sacs *(rdf : Bag), (rdf:Seq) et (rdf:Alt).*

Constructeurs d'individus

Les instances des classes RDFS sont définies en RDF, par deux types de triplets :

1. Les triplets de la forme *(i, rdf : type, C)* indiquent que *i* est une instance de la classe *C*. RDFS supporte la multi-instanciation qui permet à une instance d'appartenir à plusieurs classes même si ces classes ne sont pas liées par une relation de subsumption (spécialisation).
2. Les autres triplets, de la forme *(i, p, v)*, caractérisent l'instance *i* par la valeur *v* pour la propriété *p*.

[6] Lorsque plusieurs classes sont spécifiées, le domaine de la propriété correspond à l'intersection de ces classes.
[7] Si plusieurs classes sont utilisées, seules les instances qui appartiennent à l'intersection peuvent se voir affectées la propriété.



Ontologie (cog= "http://rdf.skikda.dz/geo/geo#")
<rdfs : class rdf : id = "cog : Personne"/>
<rdfs : class rdf : id = "cog : Document"/>
<rdfs : class rdf : id = "cog : Etudiant">
<rdfs : subclassof rdf : about= "cog : Personne"/>
<rdfs : class>
<rdf : Property rdf : id = "cog :nomPrenom"/>
<rdfs : label xml: lang="fr">nom d'usager</rdfs: lable>
<rdfs : domain rdf : about= "cog : Personne" />
<rdfs : range rdf : resource = "xsd : string" />
<rdf : Property >

Données
<cog : Document rdf : about= " http://rdf.skikda.dz/geo/doc#01"/>
<cog : Etudiant rdf : about= " http://rdf.skikda.dz/geo/pers#01"/>
<cog : nomPrenom> Samir SELLAMI</cog : nomPrenom>
<cog : aEcrit rdf : resource = "
http://rdf.skikda.dz/geo/doc#01"/>
</cog : Etudiant>

Fig. 2.2 – Exemple d'ontologie exprimée en RDF schéma.

Autres concepts

- RDFS définit la classe ressource comme classe mère de toute classe. Tout est une ressource dans le Web sémantique, sauf la notion de littéral. Et, toute classe est une sous-classe de la classe ressource.
- Les classes, les propriétés et les instances peuvent également être décrites de façon textuelle en utilisant les attributs *rdfs : label* et *rdfs : comment*, *rdfs : seeAlso*, *rdfs : isDefinedBy*.

La Figure 2.2 montre un exemple d'ontologie présentée sur la forme d'un graphe. Cette ontologie comporte les classes Personne, Etudiant (sous-classe de Personne), Document et Territoire. En dessous de cette ontologie, nous avons présenté des données de cette ontologie. Les URI des instances sont entourées par des ovales tandis que les valeurs littérales sont représentées dans des rectangles. Enfin, nous présentons un extrait de l'ontologie et des données suivant la syntaxe RDF/XML qui est une syntaxe XML pour les ontologies RDFS. Nous voyons notamment avec cette syntaxe que la propriété nom de la classe Personne représente le nom d'usage de la Personne. Cette précision est fournie par le constructeur *rdfs : label.* Notons que, comme le montre cette Figure, il est possible de préciser avec le constructeurs *rdfs : label* un attribut *xml : lang* qui renseigne sur la langue utilisée dans la description du label.

RDFS a étendu RDF par un ensemble de constructeurs permettant la définition d'ontologies sur le Web. Cependant, RDFS ne contient aucune primitive explicite permettant de décrire des équivalences conceptuelles. Les ontologies définies suivant le formalisme RDFS sont donc être composées de concepts canoniques uniquement. Dans la section suivante nous présentons le formalisme PLIB qui dans le domaine de l'ingénierie permet de définir des ontologies canoniques beaucoup plus complètes.

3.1.2 PLIB

PLIB (Parts LIBrary: séries de normes ISO 13584), initié en 1987, est un formalisme d'ontologie conçu initialement dans le cadre du domaine technique pour échanger et modéliser avec la plus grande précision possible les différentes catégories (classes) de composants industriels et leurs instances telles qu'elles sont, par exemple, décrites dans les catalogues. Pour cela, une ontologie PLIB définit de façon très fine, les catégories et les propriétés qui caractérisent les objets d'un domaine du monde réel, ainsi que les abstractions que les différentes communautés peuvent en construire [63]. Enfin, PLIB fournit des opérateurs de modularité permettant d'intégrer dans un environnement homogène et cohérent les ontologies partiellement hétérogènes définies par différentes sources.

Nous présentons ci-dessous les différents constructeurs du formalisme d'ontologie PLIB.

Constructeurs de classes

Le modèle PLIB permet de déclarer des classes et de les organiser en des hiérarchies de subsumption. Une classe PLIB peut être définie comme étant :

- une classe de définition *(item_class)* qui contient les propriétés essentielles d'une classe, c'est à dire les propriétés communes qui caractérisent les instances de la classe pour tous les acteurs qui utilisent l'ontologie ;
- une classe de représentation *(functional_model_class)* qui contient les propriétés qui n'ont de sens que par rapport à un point de vue « métier » (par exemple : le nombre d'instances qui existent en stock, pour le gestionnaire de stock, ou le taux de remise par quantité, pour le commercial) ;
- une classe de point de vue *(functional_view_ class)* qui définit la perspective dans laquelle sont définies les propriétés des classes de représentation (par exemple : gestion de stock, conditions commerciales, ...).

La hiérarchie des classes PLIB est définie par :

- la relation sémantique de subsumption nommée *is_a* qui définit une hiérarchie (simple) avec factorisation/héritage des propriétés ;
- une deuxième relation sémantique nommée *is_case_of* qui permet également d'exprimer la subsumption entre classes. Celle-ci n'est cependant pas codée par le mécanisme d'héritage. La relation sémantique *is_case_of* permet d'indiquer qu'une classe est incluse dans une autre classe (subsumption) mais qu'elle souhaite, au niveau logique, n'importer explicitement qu'une partie des propriétés de cette dernière.

Le mécanisme *is_case_of* permet de construire des ontologies modulaires qui n'importent des autres ontologies du domaine que certaines classes et pour chacune de ces classes, que le sous-ensemble des propriétés nécessaires pour l'objectif visé. Ainsi, ce mécanisme permet la définition d'ontologies autonomes qui restent toutefois articulées aux autres ontologies du domaine par subsumption. Cette articulation formelle va ainsi permettre de partager et d'échanger ce qui est commun. La relation de subsumption avec importation sélective de propriétés *is_case_of* permet aux constructeurs de redéfinir entièrement la structure des classes en fonction des besoins particuliers que vise à résoudre l'ontologie mise en œuvre. Elle permet

aussi d'assurer l'autonomie structurelle (superclasse, propriété) que temporelle (évolution éventuelle des autres ontologies du domaine) d'une ontologie. En effet, l'ontologie en cours de définition ne contient pas directement les classes importées des autres ontologies. Elle contient seulement des classes spécifiques qui lui sont propres et qui sont reliées aux autres ontologies par des relations de subsumption. Ces classes importent explicitement certaines des propriétés de la classe subsumant qui s'avèrent pertinentes pour le système en cours de construction et ceci, dans une version temporelle bien définie. De plus, PLIB ne fournissant que des constructeurs canoniques, les ontologies PLIB ne comportent que des concepts primitifs et représentent donc des modèles canoniques d'échanges.

Constructeurs de propriétés

La modèle PLIB permet également de déclarer des propriétés en tenant compte du fait que la valeur d'une propriété peut dépendre d'une, ou plusieurs autres propriétés. Une propriété PLIB peut être définie comme étant :

- une propriété autonome *(non_dependent_p_det) ;* sa valeur est indépendante de toute autre propriété ;
- un paramètre de contexte *(condition_det) ;* qui décrit le contexte dans lequel peut être situé un objet (exemple : la température environnante) ;
- une propriété dépendante *(dependent_p_det)* dont la valeur dépend de paramètres de contexte (par exemple : la longueur d'une tige métallique dépend de la température environnante) ;
- une propriété de représentation définie dans une classe de point de vue ou représentation *(representation_det) ;* La valeur d'une telle propriété peut changer sans que cela ne change l'objet décrit (exemple : le nombre d'instances existant en stock).

Toute propriété PLIB précise obligatoirement son domaine c'est-à-dire la classe dans laquelle l'affecter *(name_scope),* ainsi que son domaine de valeurs ou Co-domaine. C'est la notion de typage fort des propriétés. Ceci est différent de RDFS où domaine et Co-domaine peuvent ne pas être précisés. Le Co-domaine d'une propriété est défini par le constructeur *data_type*. Ce dernier peut être une classe *(class_instance_type),* une collection *(set_type, bag_type, list_type, array_type)* ou un autre type de données.

En pratique, le domaine de valeurs d'une propriété est souvent composé de plusieurs classes qui peuvent se trouver dans plusieurs branches de la hiérarchie des classes. Pour résoudre ce problème, PLIB propose de qualifier les propriétés de la façon suivante.

- Une propriété est définie au plus haut niveau de la hiérarchie *(is_a)* où l'on peut la définir sans ambiguïté ; elle est dite *visible* pour tout le sous arbre correspondant.
- Une propriété visible en un nœud peut y devenir applicable ; cela signifie qu'elle est rigide c'est à dire essentielle pour tout objet de la classe et pour toute sous-classe : toute instance réelle doit présenter une valeur ou une grandeur qui représente cette propriété. Ceci ne signifie pas que toutes les valeurs doivent être présentes dans la représentation informatique puisque ceci dépend uniquement des besoins d'utilisateur.

Identification de concepts

Afin de pouvoir référencer de façon non ambigüe et multilingue n'importe lequel de ces concepts (classes et propriétés), PLIB comporte un schéma d'identification universel (GUI : Globally Unique Identifier). Chaque organisation qui est la source d'une ontologie est associée à un identificateur unique (en général préexistant pour toute organisation ou établissement ; par exemple en France il peut en particulier, être construit sur les codes *SIRET «* Système d'Identification du Répertoire des Etablissements *»* ou *SIREN «* Système d'Identification du Répertoire des ENtreprises »). Le BSU (Basic Semantic Unit) dans PLIB est obtenu par concaténation d'un certain nombre d'attributs (code, version, révision, ...). Chaque source doit alors attribuer un code unique à chacune des classes qu'elle définit. Enfin le code d'une propriété doit être unique pour une classe et toutes ses sous-classes. La concaténation de ces informations permet alors d'identifier de façon unique et universelle chacun des concepts ci-dessus. C'est ce simple code, appelé un BSU, qu'il sera suffisant de référencer pour caractériser une classe ou une propriété PLIB. Ce code joue un rôle identique à l'URI de RDF/RDFS.

Types de données

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

- des types de données primitifs tels que les entiers, les réels, etc. ;
- des types de données énumérés *(non_ quantitative_code_type, non_quantitative_int_type)* ;
- des agrégats *(list_type, set_type, bag_type, array_type) ;*
- des données qualifiées, par exemple des unités ou des monnaies *(int_currency_type, int_measure_type, ...).*

Constructeurs d'individus

Notons que une ontologie décrivant les propriétés d'une classe indépendamment de toute utilisation dans un contexte particulier, les propriétés à utiliser pour décrire les instances de classe des ontologies PLIB dans tout contexte particulier sont choisies par l'utilisateur parmi les propriétés applicables. Il n'y a aucune contrainte sur les propriétés devant être utilisées. PLIB ne permet pas la multi-instanciation. A la place, PLIB offre le mécanisme d'agrégation d'instances. Pour cela, au travers des constructeurs de classes définis avant, PLIB propose de distinguer les propriétés essentielles d'une classe de celles qui dépendent d'un point de vue sur le concept représenté. Ainsi, une instance peut appartenir non seulement à sa classe de base mais également aux classes de représentation attachées à cette classe de base. Pour éviter toute redondance au niveau des valeurs des propriétés des instances, les propriétés définies sur la classe de base et sur les classes de représentation sont disjointes, sauf les propriétés définies comme communes qui permettent de réaliser la jointure.

Pour illustrer cette approche de gestion par le formalisme PLIB de la multi-instanciation, considérons par exemple la planète Mars. Cette dernière peut être décrite comme une instance de la classe Planète par les deux propriétés autonomes nom et couleur. Elle peut également être décrite suivant le point de vue d'un Système d'Information Géographique (SIG) ou encore du point de vue d'un logiciel de cartographie.

- Du point de vue d'un SIG, la planète Mars peut être modélisée comme étant un vecteur de points correspondant à un réseau de triangles obtenu par triangularisation. Cette représentation serait ensuite utilisée pour réaliser des calculs mathématiques divers tels que son volume, etc.
- Du point de vue d'un logiciel de cartographie, la planète Mars peut être vue comme une matrice d'altitudes obtenue par un balayage de sa surface. Cette représentation serait ensuite utilisée pour réaliser des aperçus graphiques ou encore pour déterminer la régularité de sa surface.

Autres concepts

PLIB permet de construire des ontologies multilingues ; c'est-à-dire où le même identifiant de concepts est associé à des descriptions dans plusieurs langues. Une ontologie, tout comme les classes et les propriétés, peut être associée à des attributs. Les attributs permettent d'associer à chaque concept une définition, un nom préféré, un nom court, des noms synonymes, une note, une remarque, une image ou encore une source d'information extérieure qui peut être un document.

Synthèse sur PLIB

Le formalisme PLIB permet de définir de manière précise les concepts du domaine modélisé. Pour cela il fournit comme RDF-Schéma des constructeurs permettant de catégoriser les informations du domaine en termes de classes et de les caractériser par des propriétés. Toutefois, PLIB dispose d'un plus grand nombre de constructeurs (de classes et de propriétés) que RDFS. Ces constructeurs permettent en particulier d'exprimer le contexte dans lequel est évaluée l'information et de représenter explicitement une unité de mesure. PLIB fournit un opérateur de modularité qui permet de définir des articulations formelles entre ontologies. Contrairement à RDFS où les ensembles des classes et des individus ne sont pas disjoints, PLIB impose la distinction entre les concepts de l'ontologie (classes et propriétés) et les individus. Également, PLIB ne supporte pas la multi-instanciation ; à la place il adopte une représentation des données suivant un mécanisme de point de vue.

Notons que comme RDFS, les constructeurs offerts par PLIB sont des constructeurs canoniques, les seuls raisonnements qui peuvent être effectuées sur les ontologies RDFS et PLIB sont donc le test de subsumption des classes de l'ontologie et, le test d'instanciation des individus. L'objectif de PLIB n'est pas de raisonner mais d'assurer la qualité des données. Dans la spécification formelle de PLIB, un grand nombre d'axiomes sont définis qui permettent d'identifier les erreurs de données. A la différence des formalismes orientés inférence que nous allons présenter ci-dessous, PLIB ne fait pas l'hypothèse que les données sont correctes de façon à pouvoir déduire de nouvelles connaissances de ces données. C'est précisément l'hypothèse sur laquelle se fondent les formalismes d'ontologies orientés inférence que nous présentons dans la section suivante.

3.2 Formalismes d'ontologies orientés inférence

Le souhait de pouvoir réaliser des inférences a conduit à la définition d'autres types de formalismes de modélisation d'ontologies. En particulier dans le domaine du Web, le pouvoir d'expression du formalisme d'ontologie RDFS a été étendu par un ensemble d'autres formalismes d'ontologies parmi lesquels OIL,

DAML-ONT, DAML-OIL et OWL. Ceci afin de concevoir des ontologies permettant de réaliser davantage de raisonnements (en plus du test subsumption) sur les ontologies et les données.

OWL (Ontology web Language), créé en 2001 par le W3C (World Wide Web Consortium), s'est imposé comme le langage standard pour représenter des ontologies dans le Web. Après une brève description des formalismes OIL, DAML-ONT, DAML+OIL, nous nous intéresserons dans la suite, au standard OWL développé tout particulièrement pour décrire les ressources du Web. Celui-ci introduit à la fois des équivalences conceptuelles et des hypothèses de correction des données permettant de réaliser des inférences.

3.2.1 DAML-ONT, OIL, DAML+OIL

Afin d'étendre le pouvoir d'expression de RDFS qui ne fournit que des mécanismes primitifs pour spécifier des classes, deux initiatives ont été lancées.

La première est une initiative américaine lancée par la DARPA (Defense Advanced Research Projects Agency), a permis la spécification du langage DAML-ONT (DARPA Agent Markup Language) [8], fondé sur RDF, et proche des langages objets.

La seconde est une initiative sponsorisée par la communauté européenne projet « On-To-Knowledge », a permis la spécification du formalisme OIL (Ontology Inference Layer) [91], basé sur XML et, offrant des primitives inspirées des logiques de description.

La fusion de ces deux langages a donné naissance au formalisme DAML+OIL [85] qui cherche à combiner toutes les caractéristiques de DAML-ONT, de OIL, de RDFS, et de RDF. DAML+OIL a servi de base pour la conception du formalisme OWL standardisé par le W3C.

3.2.2 OWL

OWL est une recommandation du W3C[9] pour indexer, à l'aide d'ontologies, les ressources du Web. L'objectif de cette approche est de permettre d'automatiser l'interprétation de la signification des ressources du Web en annotant ces ressources avec des termes ontologiques traitables par machine [17]. OWL, permet à la fois d'annoter les données et de faire certains raisonnements sur les données. Pour cela, le formalisme OWL offre en plus des constructeurs du formalisme RDFS, d'autres constructeurs qui vont permettre de réaliser des raisonnements.

Un des objectifs importants de OWL est de pouvoir assurer que les raisonnements qui peuvent être réalisés sur les ontologies soit décidables, c'est-à-dire qu'il soit possible de concevoir un algorithme qui puisse garantir de déterminer, dans un temps fini, si oui ou non les définitions fournis par une ontologie OWL peuvent être déduites d'une autre ontologie. L'impossibilité d'obtenir un formalisme à la fois compatible avec RDFS et dont les raisonnements associés soient décidables a poussé les concepteurs du formalisme d'ontologie OWL à en spécifier trois sous-formalismes de compatibilité et d'expressivité croissante mais, de décidabilité décroissante : OWL Lite, OWL DL et OWL Full.

Les influences d'OWL

La conception du langage OWL a été influencée à la fois par (1) les langages préexistants sur le Web, en particulier RDF et RDFS, (2) la logique de description et (3) les langages de frames.

[8] http://www.daml.org/

[9] World Wide Web Consortium. http://www.w3.org/

- La première et principale source d'influence d'OWL est issue de ses prédécesseurs. Parmi les impératifs de conception du langage OWL Figure la compatibilité avec RDF. Celle-ci se traduit par l'utilisation de RDF/XML comme syntaxe concrète d'OWL. OWL a également été inspiré de DAML+OIL pour sa sémantique.
- Le domaine de la logique de description (Description Logics) est la seconde source d'influence d'OWL. Elle a, d'une part, influencé le choix de spécifier la sémantique du langage par la théorie du modèle. D'autre part, les études sur la complexité menées dans ce domaine ont également orienté les choix des constructions du langage. En effet, un autre objectif important d'OWL est de pouvoir assurer qu'il est possible de lui associer un moteur d'inférence décidable basé sur ce langage.
- Enfin, la troisième source d'influence est le domaine des Frames (Frames Paradigm) qui consiste à regrouper l'ensemble des informations qui se rattachent à un même élément. Ceci a influencé la structuration de la syntaxe abstraite d'OWL.

Nous décrivons, dans ce qui suit, les constructeurs offerts par OWL Lite qui est le sous-ensemble d'OWL le moins expressif. Les limitations apportées à OWL Lite par rapport à OWL DL et OWL Full, produisent un sous-ensemble pratique et minimal des caractéristiques du formalisme OWL dont la mise en œuvre est la plus simple pour les développeurs d'outils d'inférence et dont les possibilités semblent suffisantes pour beaucoup de cas d'applications.

Constructeurs de classes

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

- une classe nommée (en utilisant le constructeur *owl : class* avec une URI) ;
- une expression de classe comme décrite ci-dessous, qui est une classe anonyme. Une expression de classe est soit :

1. Une restriction de propriété. Il existe des restrictions de propriétés locales qui restreignent le Co-domaine de certaines propriétés uniquement pour la classe où elles sont spécifiées.
   - ∃ P.C *(owl : someValuesFrom) :* au moins une des valeurs de P est une instance de la classe C.
   - ∀ P.C *(owl : allValuesFrom) :* toutes les valeurs de P sont des instances de la classe C.
2. Une restriction de cardinalité qui restreint localement le nombre de valeurs distinctes d'une propriété P. On distingue :

   (a) Les restrictions non typées qui ne précisent pas le type du Co-domaine.
   - ≥ 1 P *(owl : minCardinality) :* P possède au moins 1 valeur.
   - ≤ 1 P *(owl : maxCardinality) :* P possède au plus 1 valeur.
   - = 1 P *(owl : Cardinality) :* P possède exactement 1 valeur.

   (b) Les restrictions typées qui précisent le type du Co-domaine.
   - ≥ 1 P.C *(owl : minCardinalityQ) :* P possède au moins 1 valeur de type C.
   - ≤ 1 P.C *(owl : maxCardinalityQ) :* P possède au plus 1 valeur de type C.
   - = 1 P.C *(owl : CardinalityQ) :* P possède exactement 1 valeur de type C.

3. Un constructeur booléen de classes

- $C_1 \cap C_2 \cap ... \cap C_n$ *(owl : intersectionOf):* la sémantique de l'intersection est identique à celle de l'instanciation multiple (c est instance de l'intersection des C_j est équivalent à *c* est instance de chacune des classes C_j).

La hiérarchie des classes OWL Lite est définie par :

1. $C \subseteq D$ *(rdfs : subClassOf) :* qui équivaut à la subsumption en logique de description. Les seuls objets qui peuvent être instances de la sous-classe C sont ceux qui sont instances de la superclasse D.
2. $C \equiv D$ *(owl : EquivalentClass) :* cet axiome équivaut à $((C \subseteq D) \wedge (C \subseteq D))$.

Constructeurs de propriétés

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

1. Une propriété de type simple *(owl : datatypePropety) :* son Co-domaine est alors un type de données issu de la spécification XML Schema (string, int, real, boolean, ...).
2. Une propriété de type objet *(owl : ObjectProperty) :* son Co-domaine est une classe de l'ontologie.

Contrairement à l'approche utilisée par les langages de programmation orientés objets, l'association des propriétés aux classes se fait lors de la définition du domaine de la propriété *(rdfs : domain).* Si aucune classe n'est spécifiée, la propriété s'applique à la classe à *owl : Thing* (c'est-à-dire à l'ensemble des classes de l'ontologie) qui est la classe racine de toute ontologie OWL. De la même manière, une propriété peut préciser explicitement son Co-domaine *(rdfs : range),* sauf qu'en plus des classes, des types de données peuvent également être utilisés.

Comme pour les classes, OWL Lite définit des constructeurs pour hiérarchiser les propriétés :

1. $\subseteq$ *(rdfs : subPropertyOf) :* qui équivaut à la subsumption de propriétés (inclusion des extensions).
2. $\equiv$ (*owl : EquivalentProperty) :* qui spécifie que des propriétés ont la même extension.

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

1. *owl : TransitiveProperty :* définit une propriété transitive, comme *« seSubdiviseEn ».*
2. *owl : SymmetricProperty :* définit une propriété symétrique, comme *« estVoisinDe ».*
3. *owl : InverseOf :* définit une propriété comme étant l'inverse d'une autre propriété *« aPourPère » et « aPourEnfant ».*
4. *owl : FunctionalProperty :* la propriété ne pas avoir plus d'une valeur comme *« estNéLe ».*
5. *owl : InverseFunctional :* une unique ressource peut être associée à la valeur de cette propriété: *« APourNoSecuriteSociale ».*

Identification des concepts

Une des caractéristiques principales de OWL est son mécanisme d'identification : pour supporter la multiplicité des ontologies sur le Web, un unique identificateur doit être associé à chaque ressource (classe, instance, propriété). Pour cela, OWL utilise les URI. L'utilisation des URI permet à tout constructeur d'ontologie de référencer et d'utiliser les ressources d'autres ontologies. Néanmoins, à la différence de PLIB,

l'importation n'est pas modulaire. Les ontologies OWL doivent importer globalement toute autre ontologie dont elle référence au moins un concept.

Types de données

Les types de données supportés par OWL Lite sont :

1. Les types de données primitifs issus de la spécification XML Schema *(xsd : string, xsd : boolean,...).*
2. Le type de donnée RDFS intégré *rdfs : Literal.*

Constructeurs d'individus

Le constructeur d'individus OWL est *: owl : individual.* Les instances d'une classe sont définies en explicitant la ou les classes auxquelles elles appartiennent et en indiquant les valeurs de propriétés sous la forme d'une liste de couples (propriété, valeur). OWL permet de définir les assertions suivantes sur individus :

1. $i \in C$ *(rdf : type) :* permet de déclarer que l'individu *i* appartient à la classe *C.*
2. $< i_1, i_2 > \in$ P, $< i_1, v > \in$ P : définie la valeur d'une propriété de type objet (l'individu i_1 a comme valeur pour la propriété P l'individu i_2) et de type simple respectivement (i_1 à la valeur simple *v* pour la propriété P).
3. $i_1 \equiv i_2$ *(owl : sameAs) :* égalité entre les individus i_1 et i_2.
4. $i_1 \neq i_2$ *(owl : differentFrom) :* différence entre les individus i_1 et i_2.

Autres concepts

Une ontologie peut importer *(owl : imports)* l'ensemble des concepts définis par une autre ontologie.

Une ontologie, tout comme les classes, propriétés et instances, peuvent être annotées ; ceci permet d'associer à un tel concept un mot *(rdfs : label),* un numéro de version *(owl : versionlnfo),* un commentaire *(rdfs : comment),* des références vers d'autres concepts *(owl : seeAlso)* ou même son créateur *(owl : isDefinedBy).* Ces annotations sont néanmoins beaucoup moins nombreuses et précises que celles de PLIB. Par exemple, il n'existe pas de moyen, en OWL, de préciser l'unité d'une mesure ou de fournir un graphique.

Synthèse sur le formalisme OWL

OWL se différencie du couple RDF/RDFS en ceci qu'il apporte à RDFS, toute l'inférence basée sur l'hypothèse de correction de données. Si RDF et RDFS apportent à l'utilisateur la capacité de décrire le domaine à modéliser avec des classes et des propriétés, OWL intègre, en plus le support d'équivalences conceptuelles : comparaison des propriétés et des classes (identité, équivalence, contraire, disjonction), caractéristiques logiques des propriétés (symétrie, transitivité, …etc.), contraintes sur les propriétés (cardinalité, valeurs permises) et équivalence des termes. Ces différents constructeurs permettent de déduire de nouvelles informations (implicites) à partir de l'information préexistante (explicite).

Le formalisme OWL se décline en trois versions d'expressivité croissante OWL Lite, OWL DL et OWL Full (OWL Lite $\subset$ OWL DL $\subset$ OWL Full). Les raisonnements associés à OWL Lite et OWL DL sont décidables mais ne sont, par contre, pas compatibles avec RDFS car ils restreignent les capacités des constructeurs de ce modèle. Par exemple, OWL Lite et OWL DL ne permettent pas qu'une classe soit une

instance d'une autre classe ce qui est possible en RDFS. En conséquence, toute ontologie RDFS n'est pas forcément une ontologie OWL Lite ou OWL DL. A l'opposé OWL Full est compatible avec RDFS mais les raisonnements associés ne sont pas forcément décidables. Notons que ces niveaux évoluent avec les nouveaux besoins rencontrés par les utilisateurs. Ainsi, OWL2 ajoute à OWLl un support plus étendu des types de données permettant ainsi la définition de types de données utilisateurs. Ces extensions sont pour la plupart supportées par la plupart des outils et raisonneurs existants (Protégé 3.5[10], Fact ++, Pellet).

4. Similitudes et différences des formalismes d'ontologie

Après l'étude des langages de définitions d'ontologies (RDFS, PLIB et OWL). On remarque que ceux-ci présentent des points en commun et des différences [92].

4.1 Similitudes

Pour ce qui concerne leurs points communs, quatre points méritent d'être soulignés.

1. Ils conceptualisent l'univers à l'aide de classes et propriétés. Les classes sont organisées en hiérarchie au moyen de relations de subsumption.
2. Ils associent chaque concept des ontologies (classes, propriétés, etc.) à un identifiant permettant de le référencer à partir de n'importe quel environnement ou système, et en particulier par une autre ontologie. En PLIB, cet identifiant est unique. Il est appelé BSU (Basic Semantic Unit) et, il est fondé sur l'identification de la source de l'information ontologique. En OWL, cet identifiant est appelé URI (Uniform Resource Identifier), il est basé sur les identifiants Web, et n'est pas nécessairement unique.
3. Ils définissent des méta-attributs qui permettent de décrire textuellement et dans différentes langues la description des concepts.
4. Les autres constructeurs qu'ils offrent répondent aux besoins d'un problème précis et ont été influencés par des courants différents.

4.2 Différences

Pour ce qui concerne leurs différences, nous constatons les faits suivants :

1. Les formalismes orientés gestion et échange des données (RDFS, PLIB) définissent un langage canonique, ce qui les rend plus adaptés pour définir des bases de données (pas de redondance ou de duplication des données). De plus, les contraintes définies dans l'ontologie sont considérées comme des contraintes d'intégrité et permettront de détecter les erreurs de données.
2. Les formalismes orientés inférence (OWL) sont plus centrés sur des opérateurs de classes qui permettent de définir des classes définies (non primitives). Ces formalismes proposent un ensemble de constructeurs d'équivalence conceptuelle (union, intersection, équivalence, ...) qui peuvent être combinés pour définir de nouvelles classes, puis raisonner sur ces différentes classes. Néanmoins, ils ne permettent pas de vérifier qu'il n'y a pas d'erreurs dans les données.Ces distinctions ont amené la communauté de Web sémantique à proposer une classification des ontologies que nous présentons dans la section ci-dessous.

[10] http://protege.stanford.edu/

4.3 Le modèle en oignon

Les formalismes de modélisation d'ontologies permettent, à travers les constructeurs qu'ils offrent, de définir des ontologies. Cependant, ces ontologies ne sont pas identiques. Au sein du laboratoire LISI [11], l'ensemble des approches de modélisation ontologiques ont été plus largement étudiés, en intégrant les approches linguistiques (dont les concepts sont représentés et identifiés par des termes). Cette étude a amené à proposer une classification des ontologies suivant un modèle en couches appelé modèle en oignon [83] (voir Figure 2.3) et constitué de trois couches.

- *les Ontologies Conceptuelles Canoniques* (OCC) contiennent les ontologies qui visent à décrire des concepts et non les mots d'un langage et dont les définitions ne contiennent aucune redondance. Les OCC (Dublincore, IEC-61360-4 (International Electronic commission)) adoptent une approche de structuration de l'information en termes de classes et de propriétés et leur associent des identifiants uniques réutilisables dans différents langages ;
- *les Ontologies Conceptuelles Non Canoniques* (OCNC) contiennent les ontologies qui décrivent également des concepts mais représentent non seulement des concepts primitifs (canoniques), mais aussi des concepts définis (non canoniques), c'est-à-dire qui peuvent être construits à partir de concept primitifs et/ou d'autres concepts définis, par exemple par les constructeurs d'équivalence définis dans OWL;
- *les Ontologies Linguistiques* (OL) contiennent les ontologies qui définissent l'ensemble des termes qui apparaissent dans la description langagière d'un domaine. Outre des définitions textuelles, un certain nombre de relations linguistiques (synonyme, hyperonyme, hyponyme, etc.) sont utilisées pour capturer de façon approximative et semi-formelle les relations entre les mots. Un exemple d'OL est WordNet[12].

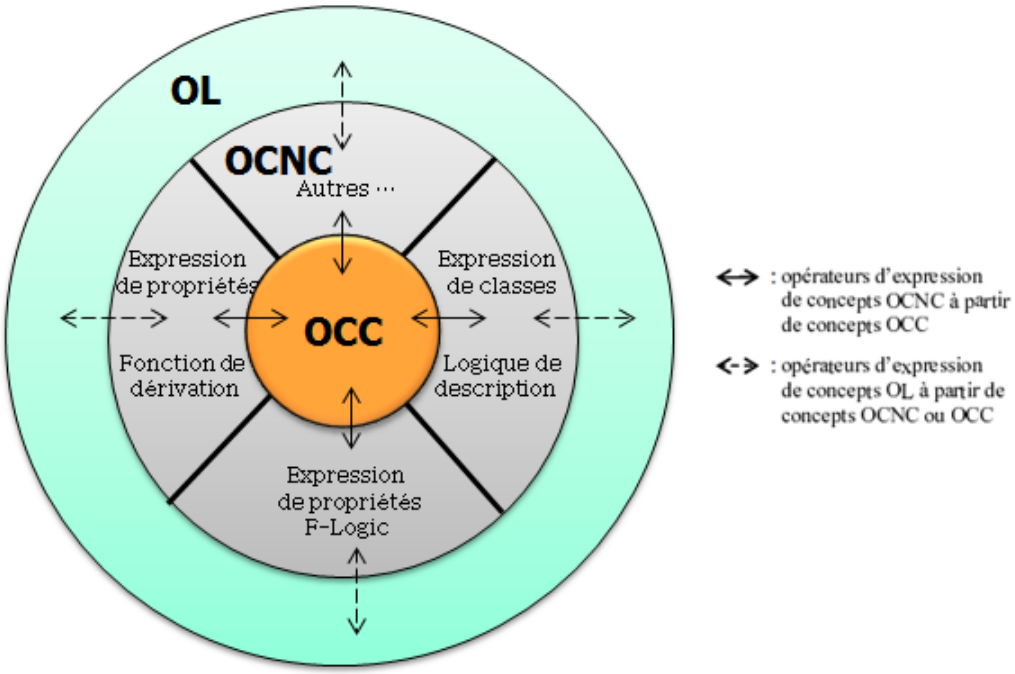


Fig. 2.3 – Le modèle en oignon.

[11] http://www.lisi.ensma.fr
[12] http://wordnet.princeton.edu/

Malgré la diversité des formalismes d'ontologies, il apparaît que tous les formalismes possèdent un noyau commun. Nous proposons d'identifier pour chaque couche du modèle en oignon, les constructions offertes par les différents formalismes d'ontologies et, de faire une comparaison de ces constructions en termes de :

- ce qu'elles permettent d'exprimer en commun,
- ce qu'elles permettent d'exprimer de manière orthogonale ou complémentaires et enfin,
- ce qu'elles ne permettent pas d'exprimer simultanément et qui nécessitera toujours un choix.

4.3.1 Ontologies Conceptuelles Canoniques

Tous les formalismes d'ontologies PLIB, RDFS et OWL offrent des constructeurs de classes et de propriétés canoniques. Ils offrent également des types de données simples (caractères, entier, réel, booléen). Ces différentes constructions permettent de caractériser de manière précise les informations du domaine à modéliser. Ainsi, ces formalismes permettent de définir avec ces constructeurs, des ontologies canoniques. Toutefois, chaque formalisme fournit également un ensemble de constructeurs spécifiques et, imposent des choix de modélisation. Certains de ces constructeurs spécifiques peuvent cohabiter sans contradiction dans le même environnement. Notons que dans l'implémentation de notre système d'intégration à base ontologique présenté dans le chapitre 4, nous avons utilisé le formalisme OWL pour décrire l'ontologie du domaine université. La motivation de ce choix est que ce langage ontologique est capable de décrire formellement la signification de la terminologie employée dans les bases données des différentes universités publiées dans leur Sites Web, et constitue le premier niveau nécessaire du Web sémantique après RDF.

Le Tableau 2.1 montre par exemple que le formalisme PLIB offre la possibilité de définir le contexte d'évaluation d'une propriété grâce au constructeur de propriété dépendante. OWL permet de définir une relation de subsumption entre les propriétés. Ces deux constructions spécifiques peuvent tout à fait être définies simultanément dans un même environnement, car elles sont orthogonales. Le formalisme PLIB adopte une approche de représentation par point de vue pour modéliser certains individus. Cette représentation ne peut cependant pas coexister simultanément avec la multi-instanciation offerte par les formalismes RDFS et OWL, ceci car PLIB ne permet pas de rattacher une instance à plusieurs classes. Il est donc nécessaire, pour cette dernière alternative de représentation des données, d'établir un choix pour chaque environnement où l'on souhaiterait définir des OCC à partir de constructions issues à la fois des formalismes PLIB, RDFS et OWL. Il en est également de même pour le constructeur de modularité offert par PLIB et la modélisation par héritage multiple disponible dans les formalismes RDFS et OWL.

Tab. 2.1 – Couche canonique des formalismes d'ontologies PLIB, RDFS et OWL.

	PLIB	**RDFS**	**OWL**
Classes			
subsumption de classes	simple / modularité	multiple	multiple
Propriétés			
typage fort	+	-	± (Lite)
dépendance entre propriété	+	-	-
domaine / Co-domaine multiple	-	+	± (Lite)
subsumption de propriétés	-	+	+
fonctionnelle	+	-	+
inverse fonctionnelle	-	-	+
cardinalité	+	-	±
Types de données			
primitifs	+	+	+
classe	+	+	+
collection	+	+	+
Individus			
identification universelle	-	+	+
multi-instanciation	-	+	+
point de vue	+	-	-
méta-modélisation	-	+	± (Full)
détection d'erreurs de données	+	-	-

4.3.2 Ontologies Conceptuelles Non-canoniques

En plus des constructeurs canoniques, certains formalismes d'ontologie offrent des constructeurs d'équivalences conceptuelles ou des prédicats constructifs qui enrichissent la couche canonique des ontologies. Le Tableau 2.2, montre les différentes constructions non-canoniques offertes par les formalismes d'ontologie PLIB et OWL DL permettent par exemple de définir des types de données énumérées. OWL permet de définir une classe comme étant l'intersection de plusieurs autres classes. Cette construction est interprétée comme la subsumption multiple, elle peut donc être représentée en utilisant l'approche modulaire définie par le formalisme PLIB en important explicitement toutes les propriétés des classes subsumantes. L'intersection de classes peut donc être implémentée dans ces différents formalismes. Les constructions telles que les caractéristiques de propriétés offertes par le formalisme d'ontologie OWL, peuvent être implantées sur la couche canoniques des ontologies PLIB. Par contre, la définition d'une classe comme étant l'union de deux classes, possible en OWL DL est impossible en PLIB où une instance appartient seulement à une classe. De même, le fait qu'une classe soit une restriction d'une autre classe ne peut pas s'exprimer directement en PLIB, même si elle n'est pas contradictoire avec les primitives de PLIB.

Tab. 2.2 – Couche non-canonique des formalismes d'ontologies PLIB, RDFS et OWL.

	PLIB	**RDFS**	**OWL**
Classes			
Intersection	±	-	+
Union	-	-	± (DL/Full)
Complément	-	-	± (DL/Full)
Énumération d'instances	±	-	± (DL/Full)
Restriction			
Co-domaine	-	-	+
cardinalité	-	-	+
égalité/inégalité de valeur	-	-	± (DL/Full)
Prédicats constructifs sur les propriétés			
réflexivité, symétrie, transitivité, ...	-	-	+
inverse	-	-	+
Types de données			
Énumérés	+	-	± (DL/full)
Unité	+	-	-
Prédicats constructifs sur la déduction de faits	-	-	+
Prédicats constructifs sur la vérification d'intégrité	+	-	-

4.3.3 Ontologies Linguistiques

Enfin, afin de permettent l'accès dans une interface langagière aux concepts des ontologies, les différents formalismes d'ontologies offrent la possibilité d'attacher à chaque concept, un ensemble d'attributs (de descripteurs) prédéfinis et ce, dans différentes langues. Le Tableau 2.3 présente les différents descripteurs disponibles dans les formalismes d'ontologies PLIB, RDFS et OWL. PLIB possède un plus grand nombre de descripteurs. Ils sont plus précis que les descripteurs OWL (qui se limite pour l'essentiel à, label et commentaire). Ils pourraient donc être utilisés pour enrichir plus spécifiquement la description des concepts OWL.

Tab. 2.3 – Couche linguistique des formalismes d'ontologies PLIB, RDFS et OWL.

	PLIB	**RDFS**	**OWL**
Classes / Propriétés			
commentaire	-	+	+
label	-	+	+
référence	-	-	+
source	-	+	+
définition	+	-	-
note	+	-	-
remarque	+	-	-
nom préféré	+	-	-
Figure	+	-	-
document de définition	+	-	-
Individus			
commentaire	-	+	+
label	-	+	+
Description multilingue	+	+	+

Le label est en général utilisé en OWL pour spécifier un nom préféré, alors que le commentaire est utilisé pour apporter davantage de précision «définition» sur un concept. Ainsi, dans tout environnement intégrant à la fois les constructions des formalismes PLIB et OWL par exemple, un choix doit être défini entre label et commentaire OWL et, nom préféré et définition PLIB.
OWL permet de rattacher à un concept une référence précisant des concepts ayant certaines relations avec un concept. Ces références peuvent être d'autres concepts de l'ontologie mais la nature de la relation n'est pas précisée. Il permet également de définir la source émettrice d'un concept. Par contre, les descriptions PLIB sont essentiellement textuelles et ne permettent pas de définir des relations avec d'autres concepts. Toutes les relations sont définies au niveau du formalisme lui-même.

5. Ontologie conceptuelle et Intégration des données

Les ontologies conceptuelles représentent un outil intéressant pour résoudre les problèmes liés à l'hétérogénéité sémantique. L'ontologie fournit une représentation explicite de la sémantique des données pouvant servir de base à la mise en correspondance des modèles différents. Elle est utilisée dans la plupart des systèmes à la fois comme un schéma global de données et comme une interface d'interrogation. L'utilisation d'une ontologie spécifique au domaine permet de ramener les concepts à un référentiel unique et de mesurer la distance et le recouvrement entre eux. Les concepts de l'ontologie sont liés aux termes des sources de données dans le méta-modèle global. Ces liens sont utilisés pour identifier les sources de données pertinentes et pour transformer les requêtes en sous-requêtes locales sur les sources originelles.

Dans cette section, nous présentons d'abord les différences entre l'ontologie conceptuelle et le modèle conceptuel. Ceci permet de rendre clair l'intérêt d'ontologie conceptuelle pour intégrer des données hétérogènes. Ensuite, nous présentons une classification des approches d'utilisation d'ontologie conceptuelle dans l'intégration de données.

5.1 Différence entre ontologie et modèle conceptuel

Le mot "Ontologie" est devenu un mot à la mode qui est utilisé souvent au lieu du mot "modèle conceptuel". En effet, un modèle conceptuel par exemple un schéma *EXPRESS* [54] développé dans le contexte de certaines normes comme ISO-10303 STEP (STandard for Exchange Product model data), est « *une spécification explicite d'une conceptualisation* ». La définition habituelle n'est pas assez précise.
Ainsi, comme remarqué par Guarino et Welty [105]

> *"Today (...ontology) is taken as nearly synonymous of knowledge engineering in AI, conceptual modeling in databases and domain modeling in OO design"*

Mais nous savons bien que la conception du modèle d'une base de données, en particulier, est réalisée selon une *approche prescriptive*. Cette approche a plusieurs implications [106]

- seules les données pertinentes pour l'application cible sont décrites ;
- les données doivent respecter les définitions et contraintes définies dans le modèle conceptuel ;
- aucun fait n'est inconnu : c'est l'hypothèse du monde fermé ;

- la conceptualisation est faite selon le point de vue des concepteurs et avec leurs conventions ;
- le modèle conceptuel est optimisé pour l'application cible.

Une telle approche engendre les problèmes d'hétérogénéité que nous avons présentés dans le chapitre 1. Par contre, la construction d'ontologie conceptuelle vise la conception d'un composant informatique générique capable de comprendre l'hétérogénéité des sources de données. Il est important donc d'expliquer la différence entre une ontologie et un modèle.

Une définition de MINSKY clarifie le rôle des modèles et souligne leur multiplicité et leur nécessité [107] : *"Pour un observateur **B**, un objet **A*** est un modèle d'un objet **A** si **B** peut utiliser **A*** pour répondre aux questions qui l'intéressent au sujet de **A**".* Cette définition souligne le caractère ternaire d'un modèle relationnel : elle dépend de l'objet (**A**) et de l'observateur (**B**), pose au sujet de **A**. Appliqué à la conception d'un modèle conceptuel, ceci montre que le modèle conceptuel dépend du contexte dans lequel le modèle conceptuel a été conçu. Le problème lorsque l'on conçoit un modèle informatique est que le contexte de modélisation est défini par les buts et l'environnement de ce système. Or, les buts et l'environnement de systèmes ne sont jamais exactement identiques. Les modèles conceptuels seront donc toujours au moins légèrement différents, et ces différences sont suffisantes pour qu'il y ait des incompatibilités entre eux.

Au contraire, le but d'une ontologie d'après la définition philosophique : *« the nature and the essence of things »* est d'être indépendant de tout contexte particulier. A la différence d'un modèle qui *prescrit* une base de données, l'ontologie décrit ce qui existe. Une ontologie sera ainsi conçue selon une *approche descriptive*. Cette approche vise à définir l'ensemble des concepts présents dans un domaine particulier et non pour une application particulière. Les données d'une application référençant une ontologie ne satisferont pas forcément l'ensemble des définitions et contraintes exprimées dans celle-ci. Ces données manquantes sont considérées comme inconnues dans le cadre de l'application. C'est *l'hypothèse du monde ouvert* [106]. Enfin, contrairement à une approche prescriptive, la conception d'une ontologie doit se faire dans un cadre *consensuel* sans contexte de conception ou alors en le définissant explicitement.

Dans ce but, il existe de plus en plus d'organismes internationaux dont l'objectif est de proposer des ontologies consensuelles (normalisées). Nous citons ci-dessous deux exemples d'ontologies normalisées

- **l'OTA** (Open Travel Alliance) [13] a été créé en 1990. Il s'agit d'un consortium qui regroupe plus de 150 organisations relatives à l'industrie du voyage. Parmi ces organisations, on trouve d'une part des fournisseurs tels que des compagnies aériennes, des hôtels, des agences de location de voitures, des compagnies d'assurance, etc., d'autre part des intermédiaires comme les GDS (Global Distribution Systems), les agences de voyages et les fournisseurs de technologie Web [19]. L'OTA vise à aider ces organisations à définir *un vocabulaire et une structure de messages commune*. En association avec la DISA (Data Interchange Standards Association), cet organisme a développé des normes de communication basées sur XML pour faciliter l'emploi du commerce électronique. L'OTA a publié environ 77 schémas XML [19] permettant de décrire des documents portant sur les besoins et les préférences des voyageurs pour des voyages d'affaires,

[13] http://www.opentravel.org/

des vacances, des voyages internationaux, des voyages en avion, des locations de voitures, des séjours en hôtel, etc. Ces schémas XML sont exploités pour créer des ontologies dans plusieurs projets concernant E-commerce ou l'intégration de données [31].

- Dans les domaines techniques, initiés en 1987 au niveau européen, l'objectif de PLIB était de permettre une modélisation informatique des catalogues de composants industriels afin de les rendre échangeables entre fournisseurs et utilisateurs. Pour réaliser ceci, le langage de modélisation *EXPRESS* (ISO 10303-11 :1994) a été utilisé pour représenter un catalogue sous forme d'instances d'un modèle. PLIB a en particulier donné lieu à un ensemble de normes ISO dans la série 13584 (Parts Library) dont les différentes parties ont été publiées entre 1996 et 2004. Parallèlement, des ontologies de domaines conformes à ce modèle ont été développées et normalisées au niveau international. La première publiée en 1998, décrit les principales catégories et propriétés de composants électroniques IEC 61360-4 :1998). Aujourd'hui plusieurs dizaines sont actuellement publiées ou en cours de développement. Certaines sont, ou visent à être, des normes (ISO 13399, ISO 10303-226, ISO 13584-501 et -511). D'autres sont développées par des consortiums industriels. L'ensemble des ontologies PLIB sont disponibles au *[www.plib.ensma.fr]*.

La section suivante présentera quelques systèmes d'intégration de données à base ontologique antérieurs.

5.2 Utilisation d'ontologies conceptuelles pour l'intégration de données

Nous présentons dans cette section comment une ontologie conceptuelle (*OC*) est utilisée dans un système d'intégration de données. L'intégration de données à base ontologique peut être divisée en trois étapes. Ces dernières sont les suivantes :

- ***la représentation de la sémantique des données*** qui vise à interpréter le sens de chaque source en l'associant à une ontologie locale. Ce processus est effectué d'une façon manuelle ou semi-automatique. En réalité, la sémantique de chaque source peut être découverte d'une façon semi-automatique [108, 109] grâce aux techniques de fouilles de données proposées dans le domaine de l'Intelligence artificielle, par exemple les techniques linguistiques [43, 110, 111], la technique de machine d'apprentissage [112], etc. Mais son résultat est approximatif, il nécessite donc la présence de l'administrateur pour valider ce résultat. Cette étape peut être éliminée si chaque source contient a priori une ontologie locale, c'est-à-dire que la sémantique d'une source est déjà sauvegardée au sein de cette source quand on la crée.
- ***l'intégration sémantique*** qui vise à intégrer les ontologies des sources. Elle consiste à établir les relations sémantiques (équivalence, subsomption) entre les concepts des ontologies. L'automatisation du processus d'intégration sémantique dépend des méthodes d'utilisation d'ontologies qui sont précisées ci-dessous.

- ***l'intégration de données*** qui vise à peupler les données dans un entrepôt pour les systèmes matérialisés ou à construire des interfaces de requêtes pour les systèmes virtuels qui fournissent une vue unique sur les données. Cette étape peut être faite par des programmes génériques qui exploitent la correspondance ontologique établie dans l'étape précédente.

Nous nous concentrons ci-dessous, sur les méthodes d'utilisation d'ontologies au sein des systèmes d'intégration. Basé sur la nature de l'étape d'intégration sémantique, nous proposons de classifier les systèmes d'intégration à base ontologique en deux classes : (1) intégration sémantique *à posteriori*, et (2) intégration sémantique *à priori*.

5.2.1 Intégration sémantique *à posteriori*

L'approche d'intégration sémantique a posteriori est caractérisée par les principes suivants

- les ontologies locales sont indépendantes ;
- l'étape d'intégration sémantique est donc effectuée d'une façon manuelle ou semi-automatique. Elle consiste à établir la correspondance entre les concepts primitifs des ontologies;
- dans ce contexte, deux structures d'intégration d'ontologies sont possibles :

1. la structure **« Multi-Ontologies »** (voir la Figure 2.4-A)

- la correspondance entre deux ontologies locales est établie directement de l'un à l'autre. Supposons qu'il existe *n* ontologies locales, il faut alors créer $[n * (n - 1)/2]$ correspondance.
- l'interface utilisateur peut être créée directement à partir d'une ontologie locale quelconque.

2. la structure **« Ontologie Partagée »** (voir la Figure 2.4-B)

- la correspondance entre deux ontologies locales est établie indirectement à travers une ontologie référencée. Cette ontologie est appelée également l'ontologie partagée du système. Une ontologie locale n'est mise en correspondance qu'avec l'ontologie partagée.
- l'ontologie partagée est soit une ontologie spécifique au système, soit une ontologie normalisée indépendante du système.
- l'interface utilisateur est créée à partir de l'ontologie partagée.

L'avantage d'une telle approche est l'indépendance des sources de données intégrées. Par contre, l'intégration sémantique n'est pas automatique. Pour la structure « multi-ontologies » le nombre de mise en correspondance entre les ontologies est important. Pour la structure "ontologie partagée", ce nombre

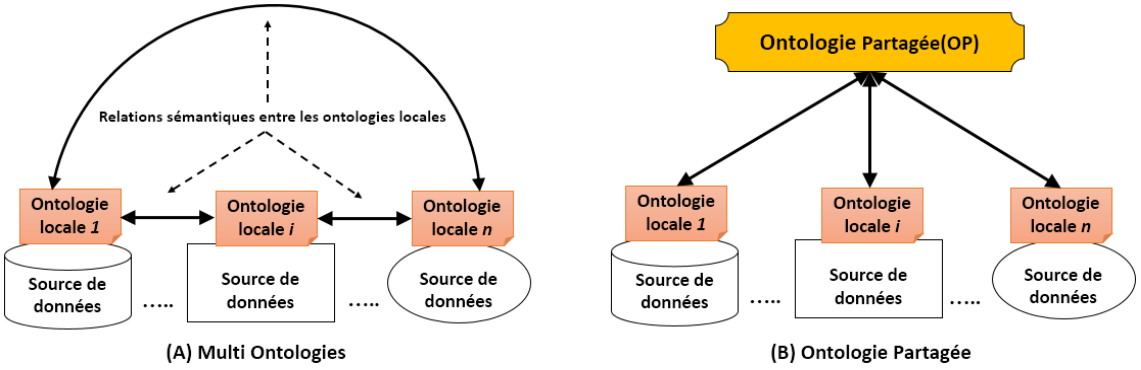


Fig. 2.4 – Utilisation d'ontologies conceptuelles dans l'approche d'intégration sémantique a posteriori.

est diminué ; et l'interface utilisateur correspond à l'ontologie partagée.

Parmi les systèmes d'intégration qui suivent ces approches, nous pouvons citer OBSERVER [29] pour la structure « multi-ontologies », et KRAFT [32] pour la structure « ontologie partagée »

- **OBSERVER** (Ontology Based System Enhanced with Relationships for Vocabulary hEterogeneity Resolution) [29]. Le but de ce projet est de fournir un système permettant l'accès transparent et uniforme à des données indépendamment de leurs localisations, leurs formats de stockage et de leurs conceptualisations. OBSERVER associe à chaque source de données une ontologie qui conceptualise son contenu. Cette association ontologie-source de données est matérialisée par des liens entre les concepts de l'ontologie et les termes de la source de données. L'intégration de plusieurs sources de données se fait par les liens sémantiques entre les concepts des ontologies. Les ontologies sont représentées dans OBSERVER en utilisant *CLASSIC* [51].
  Le système d'intégration OBSERVER (voir la Figure 2.5) se compose de groupes de sources de données. Chaque groupe possède une ontologie locale, un serveur d'ontologies et un processeur de requêtes.
  1. *l'ontologie locale :* contient les définitions des concepts locaux qui représentent la sémantique des sources de données de ce groupe.
  2. *le serveur d'ontologie :* contient les explications sur les termes utilisés dans l'ontologie locale. Il garde également les liens des concepts ontologiques vers les données (ou plutôt vers les structures de données). Cela permet de retrouver les informations à partir des sources de données et de fournir les informations nécessaires à l'administrateur pour établir les relations sémantiques entre les ontologies locales des groupes de sources différents.
  3. *le processeur de requêtes :* présente à l'utilisateur une interface lui permettant de formuler sa requête en termes des concepts présents au niveau de chaque groupe.

  Le lien entre groupes de sources est assuré par le module de gestionnaire des relations inter-ontologies. OBSERVER stocke dans ce module toutes les informations concernant les relations sémantiques (subsomption/équivalence) entre des concepts appartenant à différentes ontologies.

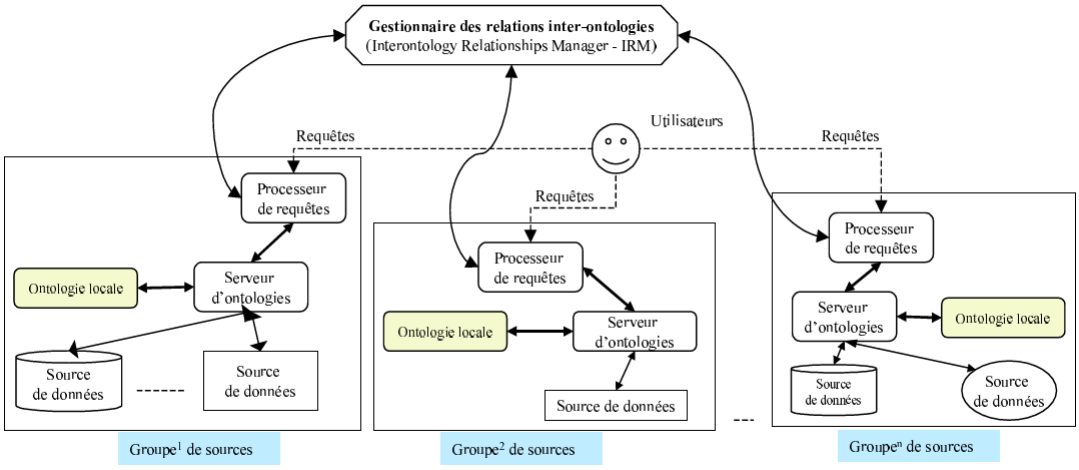


Fig. 2.5 – L'architecture de OBSERVER.

Les requêtes du système sont d'abord formulées à partir des concepts d'une ontologie choisie par l'utilisateur. Elles sont ensuite analysées par le système en utilisant les mécanismes d'inférence ontologique afin de déterminer les sources pertinentes et de transformer la requête dans le langage de la source. D'après OBSERVER, sa stratégie permet d'éviter de mettre en place une ontologie globale et de développer des ontologies spécifiques aux besoins d'utilisateurs qu'on peut extraire à partir des ontologies locales.

- **KRAFT** (Knowledge Reuse and Fusion/Transformation) [32, 113], par exemple, est un projet de recherche coopérative entre trois universités britanniques (l'Université de Aberdeen, l'Université de Cardiff et l'Université de Liverpool) avec BT (British Telecommunication). KRAFT propose une architecture d'agents dans le but d'intégration des sources de données hétérogènes. Dans KRAFT, les sources de données possèdent leurs ontologies propres. Ces ontologies sont décrites indépendamment. Elles ne sont pas obligées d'avoir un même format de représentation. Au contraire d'OBSERVER, une ontologie locale est mise uniquement en correspondance avec l'ontologie partagée du système. Cette mise en correspondance est faite d'une façon manuelle. KRAFT utilise le langage *CIF* (Constraint Interchange Format) pour spécifier la correspondance ontologique. Un système d'intégration KRAFT se compose des types d'agents ci-dessous (voir la Figure 2.6).
  1. *User Agent (UA) :* représente l'interface utilisateur.
  2. *Wrapper (W) :* représente l'interface entre une (des) source(s) de données ou une des UA et l'intérieur du système. *W* fournit des services traduisant des formats externes en un format interne (KRAFT message). *W* contient également la correspondance (en le format *CIF*) entre l'ontologie partagée et l'ontologie de la source. A l'intérieur du système KRAFT, toutes les informations sont donc homogènes.
  3. *Facilitator (F) :* s'occupe d'identifier les agents médiateurs *M* et/ou les agents *W* (correspondant aux sources de données) qui peuvent être utiles pour répondre à une requête.

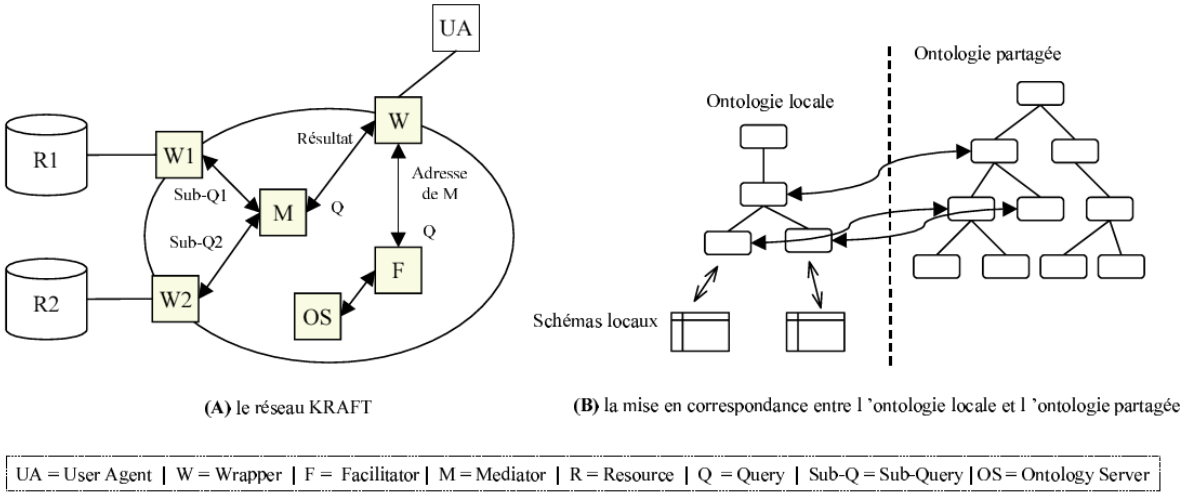


Fig. 2.6 – Architecture du système d'intégration proposée par le projet KRAFT.

4. *Médiator (M) :* se charge de répondre des requêtes complexes. Il reçoit des requêtes, puis les décompose en requêtes simples. Ces requêtes simples sont ensuite envoyées aux *W* correspondants. Pour les requêtes qu'il ne peut pas décomposer, il les signale à *F*. *F* cherche ensuite un autre *M* qui peut répondre à ces requêtes.
5. *Ontology Server (OS) :* est l'agent qui contient des informations concernant l'ontologie partagée. Il répond aux questions concernant les concepts du domaine posées par *F*.

En résumé, l'intégration de données par l'intégration sémantique *a posteriori* est proposée pour le cas où les sources de données intégrées possèdent des ontologies locales indépendantes. Une ontologie locale respecte uniquement sa source de données. Ainsi, l'intégration sémantique est faite de façon manuelle. Elle exige donc une bonne compréhension de l'administrateur du système sur chaque source. Ceci limite le passage à l'échelle du système.

5.2.2 Intégration sémantique *à priori*

L'approche d'intégration sémantique a priori est caractérisée par les principes suivants (voir la Figure 2.7)

– les administrateurs veulent une communication directe entre les sources de données intégrées. Chaque source reprend a priori des concepts dans une ontologie de domaine préexistante pour construire son ontologie locale. Cette ontologie de domaine est considérée comme l'ontologie globale du système. L'ontologie locale peut être vue comme un sous-ensemble de l'ontologie globale.

– l'intégration sémantique est donc naturellement automatique grâce aux relations sémantiques a priori entre les ontologies locales.

– l'interface utilisateur est créée à partir de l'ontologie globale.

Un système d'intégration de données suivant cette approche n'intègre que les données dont la sémantique est présentée par l'ontologie globale (OG).

Cette approche permet d'intégrer facilement une nouvelle source dans le système si la sémantique de cette source est couverte par OG. Par contre, cet aspect limite les sources de données au niveau de leurs indépendances. Parmi les projets utilisant cette méthode, nous pouvons citer SIMS [30], PICSEL [64], OntoBroker [28], BUSTER [114], COIN [6].

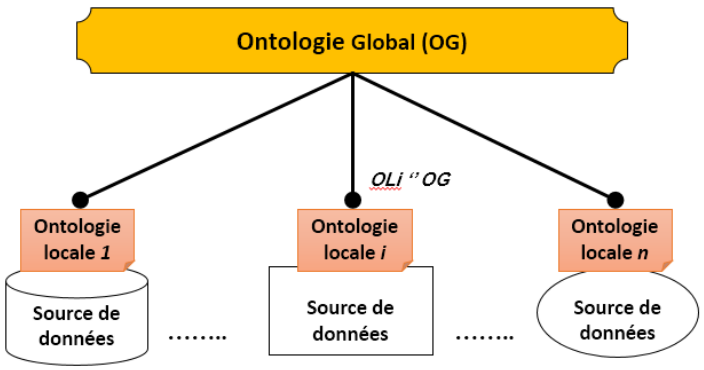


Fig. 2.7 – Utilisation d'ontologies conceptuelles dans l'approche d'intégration sémantique a priori.

- **PICSEL**, par exemple, propose une structure médiateur (voir la Figure 2.8) qui permet d'interroger des sources d'information multiples, hétérogènes et éventuellement réparties. Les systèmes médiateurs auxquels PICSEL s'intéresse regroupent un ensemble important de sources d'information XML relatives à un même domaine d'application. Dans PICSEL : « une ontologie est un élément central. Son rôle est double. D'une part, elle fournit aux utilisateurs un vocabulaire de base approprié pour formuler leurs requêtes et interroger les sources auxquelles le médiateur a donné accès. D'autre part, elle permet la description des connaissances contenues dans les sources d'information interrogeables à l'aide d'un même vocabulaire et établit, de ce fait, une connexion entre elles [64] ».

Le système de médiateur PICSEL comporte un moteur de requêtes générique et une partie de base de connaissances spécifique au domaine

1. le moteur de requêtes est conçu d'une façon générique pour être utilisable quel que soit le domaine d'application.
2. la base de connaissances est spécifique au domaine appliqué. Elle se compose d'une ontologie du domaine et des ontologies locales. L'ontologie du domaine préexistante modélise le domaine d'application et fournit ainsi, un vocabulaire structuré servant de support à l'expression des requêtes. Les ontologies locales décrivant le contenu des sources d'information sont formées a priori à partir des termes de l'ontologie du domaine.

PICSEL utilise *CARIN* [64] comme le langage de représentation de connaissances. Ce langage est un formalisme qui combine dans un cadre logique et homogène un langage de règles et un langage de classes. L'ontologie du domaine est représentée dans le formalisme *CARIN-ALN*, à l'aide de deux composantes : la composante terminologique et la composante déductive [19].

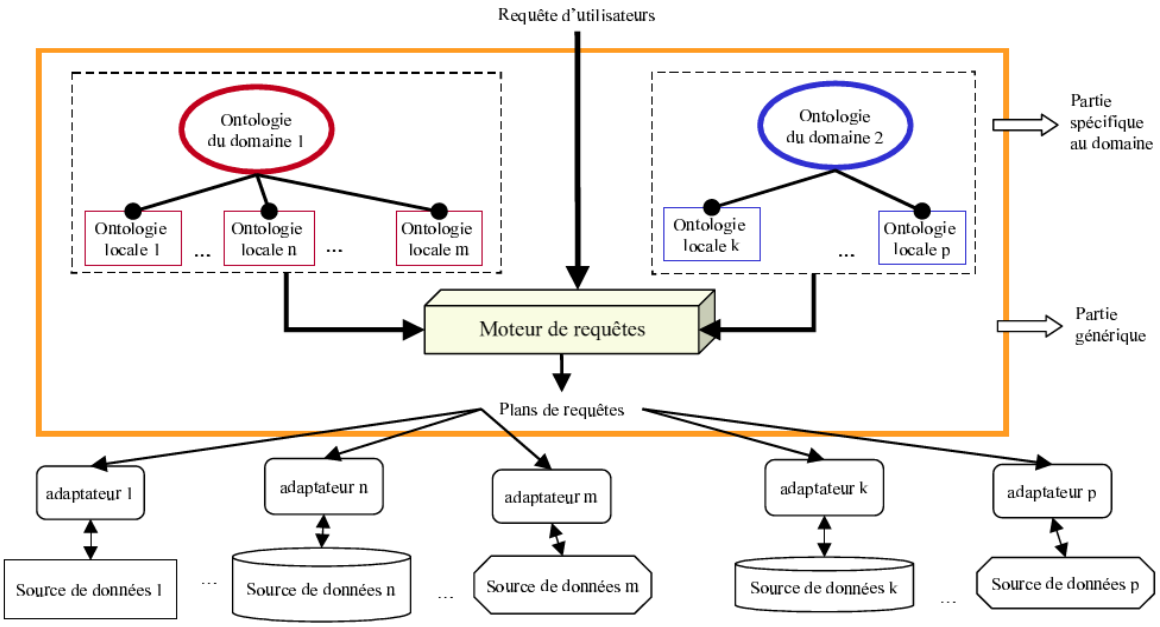


Fig. 2.8 – Architecture du système médiateur PICSEL.

PICSEL possède également un langage de vues [19] et un langage de requêtes permettant d'exprimer, en termes de l'ontologie du domaine, respectivement, le contenu des sources et les requêtes des utilisateurs.

L'ontologie globale (l'Ontologie du domaine) dans PICSEL est créée à travers l'ONTOMEDIA (Ontologie pour un MEDIAteur) [19]. Le système ONTOMEDIA permet de construire de façon semi-automatique une ontologie du domaine. Il s'agit d'un analyseur syntaxique dont la fonctionnalité consiste à appliquer un ensemble d'heuristiques pour identifier, parmi un ensemble de DTDs relatives à un domaine, les termes qui vont composer l'ontologie de ce domaine, puis de les organiser entre eux.

Un autre exemple est **OntoBroker** [65]. L'approche menée dans ce projet consiste à annoter les documents HTML du Web par les concepts d'une ontologie, de collecter les informations et les stocker d'une façon centralisée dans une base de connaissance qui sera interrogée par un langage de requêtes dédiée. L'idée centrale dans cette approche est d'utiliser une ontologie préexistante pour décrire la base de connaissance, formuler les requêtes, ajouter de la sémantique aux documents HTML du Web et pour la conception de DTD de document XML. L'ontologie est utilisée par un moteur d'inférence pour inférer de nouveaux faits dans la base de connaissance. Le langage de requêtes utilise l'ontologie pour assister l'utilisateur dans le choix des termes de sa requête.

En résumé, l'intégration de données sémantique à priori permet *une automatisation effective* pour autant que chaque source référence exactement la même ontologie, sans possibilité d'extension ou d'adaptation. L'ontologie partagée est en fait un schéma global, et, en conséquence, chaque source locale à moins d'autonomie.

Dans le chapitre suivant, nous proposons une solution pour l'intégration de sources de données relationnelles hétérogènes, fondée sur la médiation et les ontologies pour résoudre les conflits sémantiques et syntaxiques.

6. Conclusion

Nous avons présenté, dans ce chapitre, les deux principales familles de formalismes d'ontologie : OWL et PLIB. Ces deux formalismes sont différents tant du point de vue de leur domaine cible initial que du point de vue de leur objectif opérationnel.

Du point de vue domaine cible, OWL vise le domaine du langage dans la mesure où il est dérivé des logiques terminologiques (ou logique de description). PLIB, quant à lui, a été développé pour modéliser les informations techniques. Du point de vue opérationnel, PLIB a été développé pour l'échange d'information industriel. Quant à OWL, il a été développé pour permettre de faire des inférences. Ceci a amené à le subdiviser en différents sous-formalismes selon la décidabilité du raisonnement.

L'analyse comparative des formalismes d'ontologie consiste à distinguer (1) leur partie canonique, qui leur permet de décrire de façon unique et non ambiguë les concepts, (2) leur partie non-canonique, qui permet de définir des modélisations équivalentes à la modélisation canonique et (3) leur partie linguistique qui permet d'enrichir les modélisations formelles, faites par les modèles précédents, par des informations textuelles destinées à clarifier ce que chaque concept dénote. En utilisant cette classification, nous avons pu distinguer, pour chacun des niveaux, les constructions identiques, les constructions orthogonales, et les constructions incompatibles entre lesquelles il faudrait choisir.

Parmi les approches d'intégration de données, l'utilisation d'ontologies conceptuelles apparaît comme la seule approche assurant l'automatisation du processus d'intégration sémantique de données. Nous avons discuté les points communs et les différences principales entre les modèles d'ontologies conceptuelles. Cette présentation synthétique a montré que si tous les modèles d'ontologies partagent les mêmes concepts fondamentaux, chacun possède des opérateurs spécifiques adaptés aux problèmes qu'ils visent à résoudre.

L'utilisation d'ontologies conceptuelles pour développer des systèmes d'intégration est récente, mais devient populaire. Nous avons présenté comment les approches existantes utilisent des ontologies conceptuelles pour résoudre les conflits sémantiques de données. Nous classifions ces approches dans deux catégories : (1) l'intégration sémantique à posteriori, et (2) l'intégration sémantique à priori. La première approche permet de garder l'indépendance de chaque source de données, mais l'intégration sémantique est faite manuellement. Au contraire, la deuxième approche n'intègre que les sources de données dont la sémantique est représentée à priori en connexion avec une ontologie de domaine, et en conséquence son processus d'intégration sémantique est automatique. Dans le chapitre suivant, nous proposons une approche à base ontologique et une architecture système pour l'intégration de sources données hétérogènes et autonomes.

Chapitre 3
Approche d'intégration à base ontologique proposée

« Il y a deux catégories de chercheurs dont les uns ne seraient que des manœuvres, tandis que les autres auraient pour mission d'inventer.

L'invention doit être partout, jusque dans les plus humbles recherches de faits, jusque dans l'expérience la plus simple.

Là où il n'y a pas un effort personnel même original, il n'y a pas un moment de sciences. »

H. Bergson

Chapitre III

Approche d'intégration à base ontologique proposée

Sommaire

1 Introduction 74
2 Approche d'intégration de données 75
2.1 Approche de notre système de médiation 75
2.2 Ontologie de domaine 79
2.3 Modèle de représentation uniforme de l'ontologie de domaine 79
2.4 Description des sources de données à intégrer 82
3 Architecture du système 83
3.1 Module de création des requêtes 84
3.1.1 Sélection de l'ontologie de domaine 84
3.1.2 Création de requête 84
3.2 Serveur d'ontologie et métadonnées des sources 84
3.2.1 Information sur l'ontologie de domaine 84
3.2.2 Information sur les sources de données 85
3.2.3 Choix de source de données locale 85
3.3 Module de mapping sémantique 85
3.4 Module de traitement des requêtes 86
3.4.1 La reformulation de la requête de l'utilisateur 86
3.4.2 Création du plan de requête 86
3.5 Module d'extraction des données 87
3.5.1 Traduction de requête de l'utilisateur au langage de requête de l'adaptateur 87
3.5.2 Extraction des données 87
3.6 Module de corrélation des données 87
3.6.1 Conversion des valeurs de données 87
3.6.2 Corrélation des réponses 87
3.6.3 Représentation de réponse 87
4 Création de requête utilisateur 88
5 Les différents cas de mapping 89
6 Schéma de mapping 91
6.1 Exemple de mapping 92
7 Discussion 95
8 Conclusion 96

1. Introduction

On assiste depuis quelques années à l'émergence de nouvelles applications qui ont besoin de partager des informations entre différents systèmes. C'est le cas du e-gouvernement, du e-learning, du e-commerce, de la bio-informatique ou encore des bibliothèques électroniques. Or, dans ce contexte, les systèmes d'informations, conçus et développés par des organisations différentes, constituent généralement des sources de données autonomes, hétérogènes et distribuées. L'accès, la récupération et l'utilisation de ces sources de données imposent une nécessité pour que les données soient intégrées en offrant aux utilisateurs un point d'accès unique aux données.

Nous proposons une solution pour l'intégration de sources de données structurées hétérogènes, fondée sur la médiation et les ontologies pour résoudre les conflits sémantiques et syntaxiques. Nous visons l'hétérogénéité sémantique aussi bien au niveau des schémas (structures) qu'au niveau du contenu (données), et ceci, tout en préservant l'autonomie des sources de données. La résolution des conflits sémantiques entre les termes de la requête utilisateur et les termes de sources de données est le principal problème à traiter.

Dans notre approche proposée:

- Nous utilisons une ontologie de domaine pour créer la requête de l'utilisateur. L'ontologie de domaine est mise au point par des experts du domaine et se compose de tous les termes et les concepts existants dans un domaine spécifique. Pour sa requête, l'utilisateur est obligé d'utiliser les termes de l'ontologie de domaine avec laquelle il doit être familier. Pour cela, nous proposons un modèle de représentation uniforme spécifique pour l'ontologie de domaine et une interface graphique adaptée à l'interaction de l'utilisateur avec l'ontologie de domaine afin de poser des requêtes.
- Nous proposons un mécanisme de mapping entre les métadonnées des schémas locaux et le modèle de représentation uniforme de l'ontologie de domaine. Le problème majeur dans les systèmes d'intégration de données à base l'ontologique est de détecter les similarités entre les schémas locaux et d'effectuer le mapping sémantique.

Dans ce chapitre, nous détaillons notre approche d'intégration à base ontologique de sources de données hétérogènes structurées relationnelles. Puis, nous présentons l'architecture de notre système d'intégration en illustrant les différentes couches et modules du système. La section 4 montre la syntaxe et la façon dont le système crée les requêtes des utilisateurs. Les différents cas du mapping entre les schémas de bases de données relationnelles vers l'ontologie de domaine qui sont prévus pour être couverts par notre approche sont exposés dans la section 5. Nous présentons ensuite, dans la section 6 notre mécanisme de mapping et le schéma de stockage des mises en correspondance en l'appliquant sur un exemple explicatif. Une analyse et étude critique de notre approche d'intégration à base ontologique est présentée dans la section 7. Le chapitre s'achève par une conclusion dans la section 8.

2. Approche d'intégration de données

Dans l'intégration des sources de données, les principaux problèmes rencontrés sont:

- Sélection des sources de données cibles.
- Résolution des conflits d'hétérogénéité.
- Traduction des requêtes utilisateurs au langage de requête des sources de données.
- Extraction des données.
- Transformation de données et leurs présentations au format commun de l'utilisateur.

Dans cette section, nous présentons notre solution à base ontologique pour l'intégration des sources de données. Le système d'intégration de données proposé est basé sur une approche médiateur suivant la technique Local-as-View (LaV) pour le sens de mise en correspondances, et une ontologie de domaine unique.

2.1 Approche de notre système de médiation

Notre approche d'intégration à base ontologique proposé dans [86] est illustrée dans la Figure 3.1. Cette approche utilise une ontologie pour la résolution des conflits sémantiques entre les schémas locaux. Nous utilisons une ontologie spécifique au domaine pour la création de requêtes des utilisateurs. Il existe une ontologie de domaine spécifique pour chaque domaine d'application qui englobe la définition sémantique des concepts qui sont nécessaires pour fournir les requêtes des utilisateurs dans ce domaine d'application particulier. L'ontologie de domaine est modélisée dans un modèle de représentation uniforme interne au système. L'utilisateur peut naviguer dans l'ontologie de domaine et choisir les termes de sa requête, ensuite le système crée la requête de l'utilisateur. Nous supposons que chaque source de données contient des métadonnées qui la décrivent.

Suite à la création de la requête de l'utilisateur, le système choisit une source de données locale et les termes de la requête de l'utilisateur sont mis en correspondance avec ceux de la source de données. Ensuite, la requête est réécrite en utilisant les termes de la source de données locale.

Ce système utilise des ontologies pour résoudre les conflits de schémas sémantiques entre les sources de données. Le système proposé résout l'hétérogénéité sémantique des schémas entre les sources de données et la requête de l'utilisateur grâce au mapping sémantique entre l'ontologie de domaine et les termes des schémas locaux.

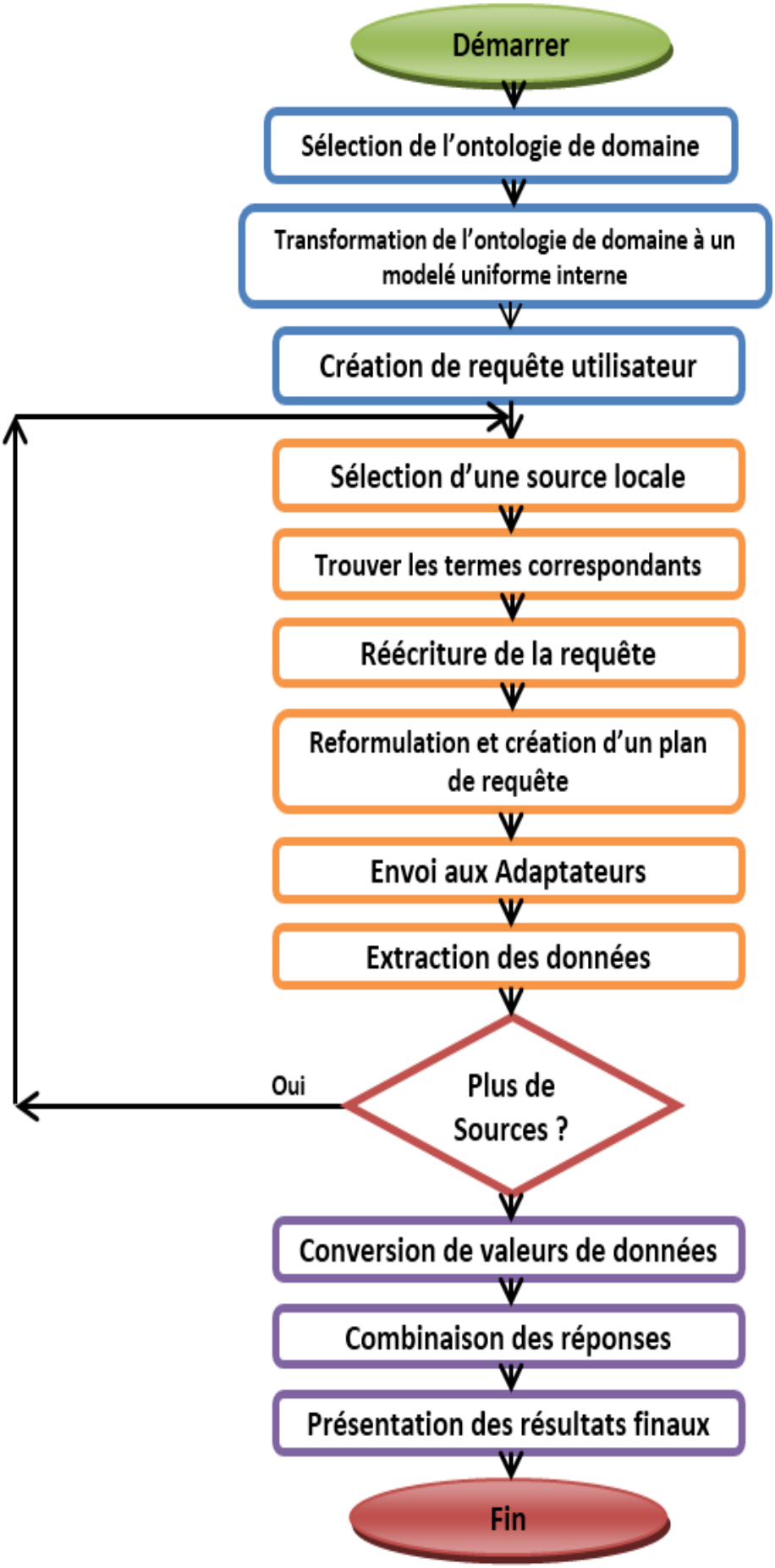


Fig. 3.1 – Approche de notre système de médiation.

La requête utilisateur réécrite précédemment est envoyée au module de traitement de requête pour sa reformulation et la création du plan de requête. Les sous-requêtes provenant du processus de reformulation sont converties aux langages de requête des sources de données, puis les réponses à ces sous-requêtes sont extraites des sources de données correspondantes par les adaptateurs. Chaque adaptateur reconnaît la structure de la source de données sous-jacente et extrait les données relatives à ses sous-requêtes et présente les données extraites dans un format commun. Grâce aux adaptateurs, le système résout les conflits structurels entre les sources de données. Les données extraites obtenues par les adaptateurs sont envoyées au convertisseur.

Le convertisseur résout les conflits d'hétérogénéité de valeur entre les valeurs de données. Le convertisseur exploite des fonctions de conversion et des règles pour résoudre les hétérogénéités entre deux valeurs de données (telles que les unités, types de données, des fonctions de conversion de format de valeur. etc.). Enfin, les opérations de plan de requête sont exécutées et les réponses définitives sont exportées au format de préférence de l'utilisateur par l'exportateur. La résolution des conflits sémantiques entre les termes de requêtes de l'utilisateur et les termes des schémas locaux est un enjeu principal dans ce système. Ce dernier résout ce problème en utilisant un mapping sémantique entre l'ontologie de domaine et les termes des sources de données locales stockées dans un schéma relationnel de mapping. Par exemple, la table "People" dans l'exemple de source de données 1 (illustré dans Figure 3.3) est mappée au concept «Staff» dans l'exemple de l'ontologie de domaine (comme le montre la Figure 3.2), un autre exemple la table «Course» est mappée au concept « Program».

Notre système met en œuvre une approche fondée sur les requêtes pour l'extraction et l'intégration de l'information, à partir de sources de données hétérogènes et distribués. Le processus d'extraction et d'intégration dans le système proposé se compose de dix grandes tâches comme suit :

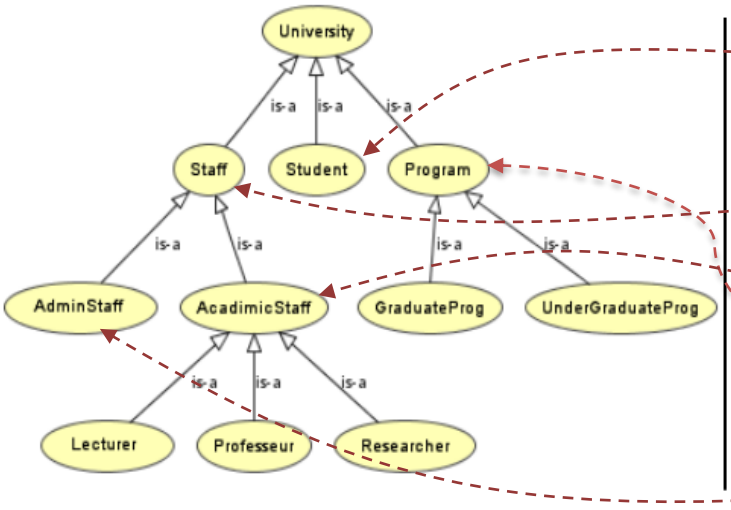


Fig. 3.2 – Une partie hiérarchique des concepts de l'ontologie de domaine (Exemple d'une ontologie de domaine).

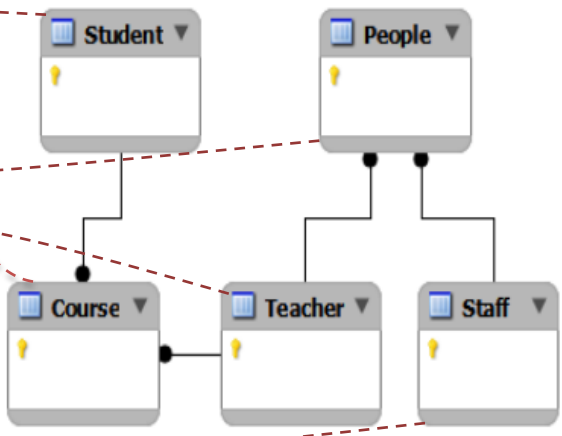


Fig. 3.3 – Une partie des tables de source de donnée 1 (Exemple de source de donnée).

1. Transformation de l'ontologie de domaine à un modèle de représentation uniforme interne ;
2. Création de requête de l'utilisateur ;
3. Détermination des sources de données sous-jacentes avec le domaine de la requête ;
4. Mapping sémantique entre les termes de la requête et les termes des sources de données locales connexes ;
5. Réécriture de la requête utilisateur avec les termes correspondants des sources de données locales ;
6. Reformulation de la requête et la création d'un plan de requête;
7. Traduction des sous-requêtes aux langages de requête des adaptateurs ;
8. Extraction des données ;
9. Conversion de valeurs de données ;
10. Exportation des données et leur présentation au format de l'utilisateur ;

Notre système d'intégration de données proposé couvre les niveaux d'abstraction des conflits d'hétérogénéité entre les sources de données locales. Le système applique :

- ***Une ontologie*** de domaine et un schéma de mapping relationnel entre les termes de chaque source et les concepts de l'ontologie de domaine, comme une solution pour résoudre les hétérogénéités sémantiques des schémas (comme le montre la Figure 3.4);
- ***Adaptateur*** comme solution pour résoudre les hétérogénéités syntaxiques des modèles de données ;
- ***Convertisseur*** comme solution pour résoudre les hétérogénéités de valeurs de données ;

Le système d'intégration de données proposé est extensible à n'importe quel domaine par l'ajout d'une ontologie de domaine spécifique au système. Cela signifie que pour pouvoir utiliser le système dans n'importe quel domaine d'application, nous devons développer et ajouter une ontologie de domaine (correspondant au domaine d'application) au système.

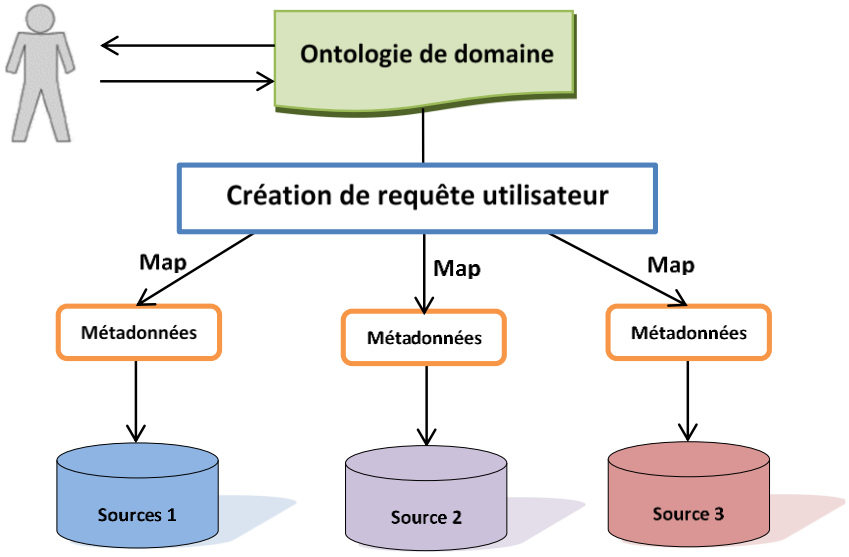


Fig. 3.4 – Architecture d'ontologie proposée pour résoudre les conflits sémantiques.

2.2 Ontologie de domaine

Notre système utilise les ontologies comme solution pour la résolution des hétérogénéités sémantiques entre les sources de données.

Une ontologie de domaine est créée pour un domaine spécifique. Par exemple, si on considère que le système a été développé pour le domaine de l'éducation supérieure (université), donc une ontologie spécifique à l'université sera conçue et créée pour le système. Les utilisateurs du système sont limités à choisir leurs termes de la requête seulement à partir de termes de l'ontologie du domaine, or, ils ne sont pas libres d'utiliser d'autres termes qu'ils préfèrent éventuellement. C'est-à-dire, Les termes utilisés dans les requêtes des utilisateurs sont exclusifs à ceux de l'ontologie du domaine.

Nous supposons que toute source de données possède des métadonnées sous-jacentes. Par conséquent, et afin de résoudre les conflits sémantiques entre les termes de la requête de l'utilisateur (choisis parmi les termes de l'ontologie de domaine) et les termes des sources de données locales, un mapping sémantique est nécessaire. Ce mapping sémantique concerne les termes similaires de l'ontologie de domaine et les sources locales en précisant la correspondance entre eux.

2.3 Modèle de représentation uniforme de l'ontologie de domaine

Le module de mapping du système reconnait des termes similaires ainsi que des correspondances entre les sources de données locales et l'ontologie de domaine, puis crée des mapping entre ces derniers. En effet, les ontologies de domaine sur le Web ont été formalisées dans différents modèles et codées dans différents formalismes (OWL, RDFS, PLIB, XML etc.) comme on a pu le constater dans le chapitre précédent.

Afin de comparer et de trouver des termes similaires entre l'ontologie de domaine et les sources de données locales, le système doit représenter l'ontologie domaine dans un modèle de représentation uniforme interne. Nous proposons un modèle de représentation uniforme des ontologies. Ce modèle de représentation est général, de sorte que n'importe quel modèle de représentation d'ontologie peut être transformé.

Définition 1 : T = (C, A, R,), chaque élément d'ontologie (terme) est l'une des entités suivantes :

- C : concept ou une instance d'un concept.
- A : attribut d'un concept.
- R : relation entre les concepts.

Par exemple "student" (concept), "age" (attribut), "master-student" (il est considéré comme sous-concept de "student" dans notre modèle), "attend" (relation entre "student" et "class") et "<20" (plage de valeurs de "max-credit-cours" une contrainte sur le concept "student") : sont des éléments (termes) de l'ontologie université.

Définition 2 : C = (Nom, Synonymes), chaque concept est défini par son nom et un ensemble de ses synonymes.

Définition 3 : A = (Nom, Synonymes Domaine, Co-domaine), chaque attribut est défini par un nom et un ensemble de synonymes, son domaine et le Co-domaine.

Définition 4 : R = (Nom, Synonymes, Domaine, Co-domaine), chaque relation est définie par un nom, un ensemble de synonymes, son domaine et le Co-domaine.

Définition 6 : O = (G, G '), chaque ontologie est représentée par deux graphes.

Définition 7 : G= (N, E), N = <C>, E = <is-a>, G est un graphe dirigé acyclique se compose de nœuds et d'arêtes. Chaque nœud est un concept (ou instance d'un concept). Chaque arête est "*is-a*" relation qui montre la subsumption entre les nœuds ; G est un modèle hiérarchique de l'ontologie. Chaque nœud a un père et ne peut n'avoir aucun, ou plusieurs nœuds enfants. Si un nœud a deux pères, le modèle résout ce problème avec la répétition nœud enfant pour chacun de ses pères.

Définition 8 : G'= (N, E'), N = <C, V>, E'= <R>, G' est un graphe cyclique qui se compose de nœuds et d'arêtes. Chaque nœud est un concept (ou une instance d'un concept) ou une valeur. Chaque arête est une relation entre deux nœuds, qui montre la relation entre les concepts. G' est un modèle de relation entre les concepts de l'ontologie.

Dans le modèle de représentation uniforme, tous les éléments (Concepts, Attributs, Relations) sont des chaînes de caractères (String). Notre modèle de représentation de l'ontologie de domaine est général, alors notre approche proposée qui utilise cette formalisation fonctionnera avec tous les langages et formalismes de représentation de l'ontologie (OWL, RDF, RDFS, XML, PLIB etc.). Nous devons transformer l'ontologie de domaine au modèle de représentation uniforme. Ce modèle de représentation constitue le principal endroit pour avoir l'information sur ontologie de domaine et qui va nous permettre de construire la requête utilisateur par la suite.

Nous utilisons une structure de tables relationnelles pour stocker l'ontologie de domaine et l'implémenter dans un SGBD tel que MySQL[14] (MySQL 5.1). Nous présentons les tables nécessaires pour le stockage de l'ontologie sous forme de modèle de représentation uniforme comme illustré dans le modèle relationnel logique de la Figure 3.5 :

[14] http://www.mysql.com/

- La table *Ontology* est utilisée pour le stockage des ontologies de domaine et leurs noms. Ontology-ID est la clé primaire de cette table.
- La table *Concepts* est utilisée pour le stockage de tous les concepts, leurs synonymes et définitions (commentaire). Concept-ID est la clé primaire de cette table. Ontology-ID est une clé étrangère qui référence la table ONTOLOGY.
- La table *Attributes* est utilisée pour le stockage de tous les attributs, leurs synonymes et commentaires dans une ontologie ainsi que le domaine et le Co-domaine (Range) de l'attribut. Attribut-ID est une clé primaire dans cette table. Domaine-ID est une clé étrangère qui référence la table CONCEPT.
- La table *Relations* est utilisée pour le stockage des relations existantes et leurs synonymes dans une ontologie. Cette table est utilisée pour stocker le modèle de relation entre concepts de l'ontologie (graphe G'). Relation-ID est la clé primaire de cette table, Domain-ID et Range-ID sont des clés étrangères qui référencent la table CONCEPT.
- La table *Hierarchy* est utilisée pour le stockage de modèle de la hiérarchie de l'ontologie (G graphe). Concept-ID est la clé primaire de cette table. Concept-ID et Father-ID référence la table CONCEPT.

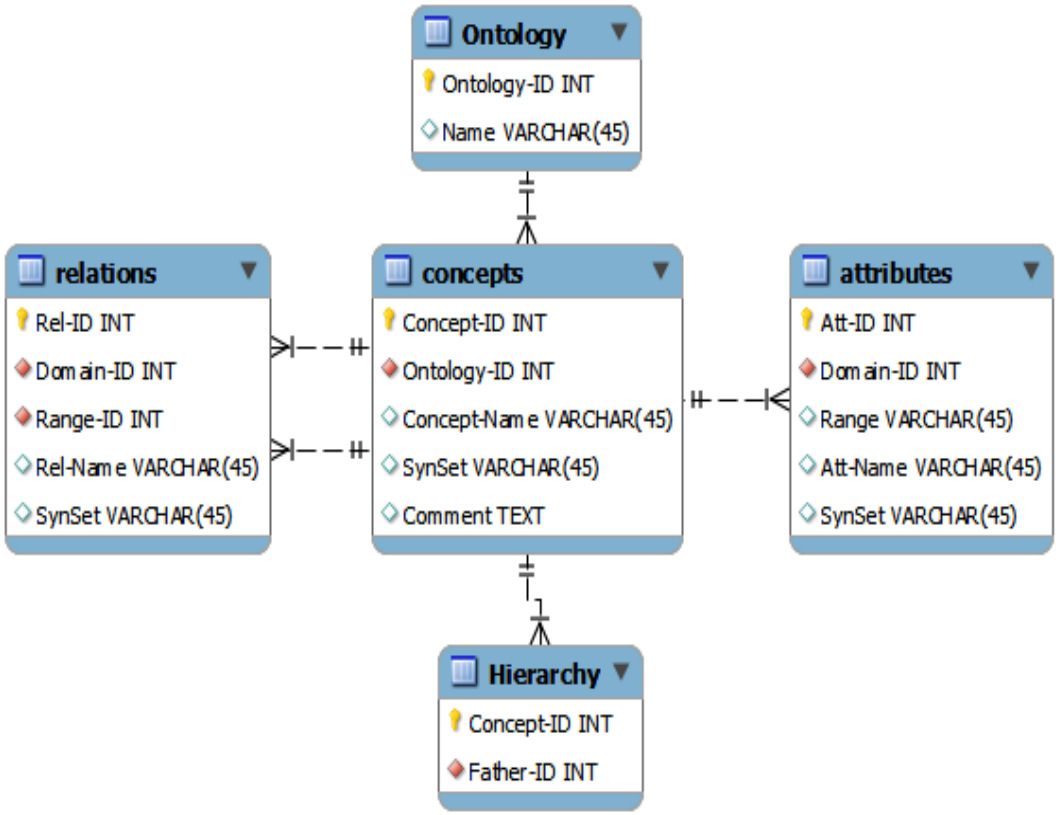


Fig. 3.5 – Modèle relationnel interne de représentation de l'ontologie de domaine.

2.4 Description des sources de données à intégrer

Dans un processus d'intégration, les sources de données à intégrer peuvent être de nature structurée, semi structurée ou bien non structurée. Dans notre approche les données sont de type structuré du fait qu'on s'intéresse à des bases de données relationnelles. Les bases de données sont structurées par leur modèle conceptuel. Ceci n'infirme pas le fait qu'elles soient hétérogènes, autonomes et réparties.

En effet, les sources de données sont conçues indépendamment (par différents concepteurs) à des fins applicatives distinctes. Il peut y avoir de différents points de vue sur le même concept, à cause des distinctes interprétations d'objets du monde réel. Du fait que les sources ont le même type de structure de données (table relationnelle), l'hétérogénéité structurelle n'est plus une difficulté majeure pour l'intégration des sources. Une multitude de travaux de recherche s'est orientée vers cet axe. L'hétérogénéité sémantique, par contre, présente un défi majeur dans le processus d'intégration. Cependant, afin de la maitriser et mener à bien le processus de mapping, nous proposons un modèle pour le stockage des métadonnées pour les différentes sources de données locales.

Nous utilisons une structure de tables relationnelles pour stocker les métadonnées sur les différentes sources de données locales et l'implémenter dans un SGBD tel que MySQL (MySQL 5.1). Ce dernier se compose de trois tables, comme illustré dans le modèle relationnel logique de la Figure 3.6.

- La table *Sources* est utilisée pour le stockage de noms des sources de données locales dans le système d'intégration. Source-ID est la clé primaire de cette table.
- La table *Tables* est utilisée pour le stockage de toutes les tables des sources avec leurs noms. Table-ID est une clé primaire dans cette table. Source-ID est une clé étrangère qui référence la table SOURCES.
- La table *Columns* est utilisée pour le stockage de tous les Attributs des tables et leurs noms. Column-ID est une clé primaire dans cette table. Table-ID est une clé étrangère qui référence la table TABLES. Le champ Constraint-Type est utilisé pour stocker le type de contrainte d'intégrité de la colonne.

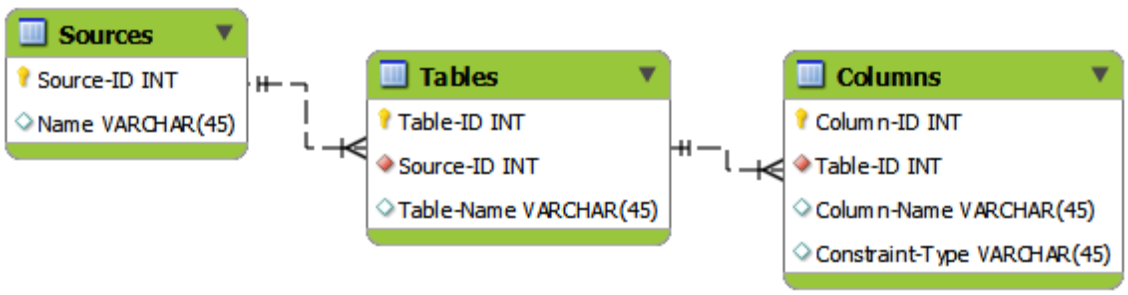


Fig. 3.6 – Modèle relationnel logique des métadonnées des sources de données locales.

3. Architecture du système

Les éléments de l'architecture du système sont illustrés dans la Figure 3.7. Notre système d'intégration de données proposé dans [90] met en œuvre une approche à base de requêtes pour l'extraction et l'intégration de l'information, à partir de sources de données relationnelles hétérogènes et distribuées. Le système possède une architecture à trois couches comme suit :

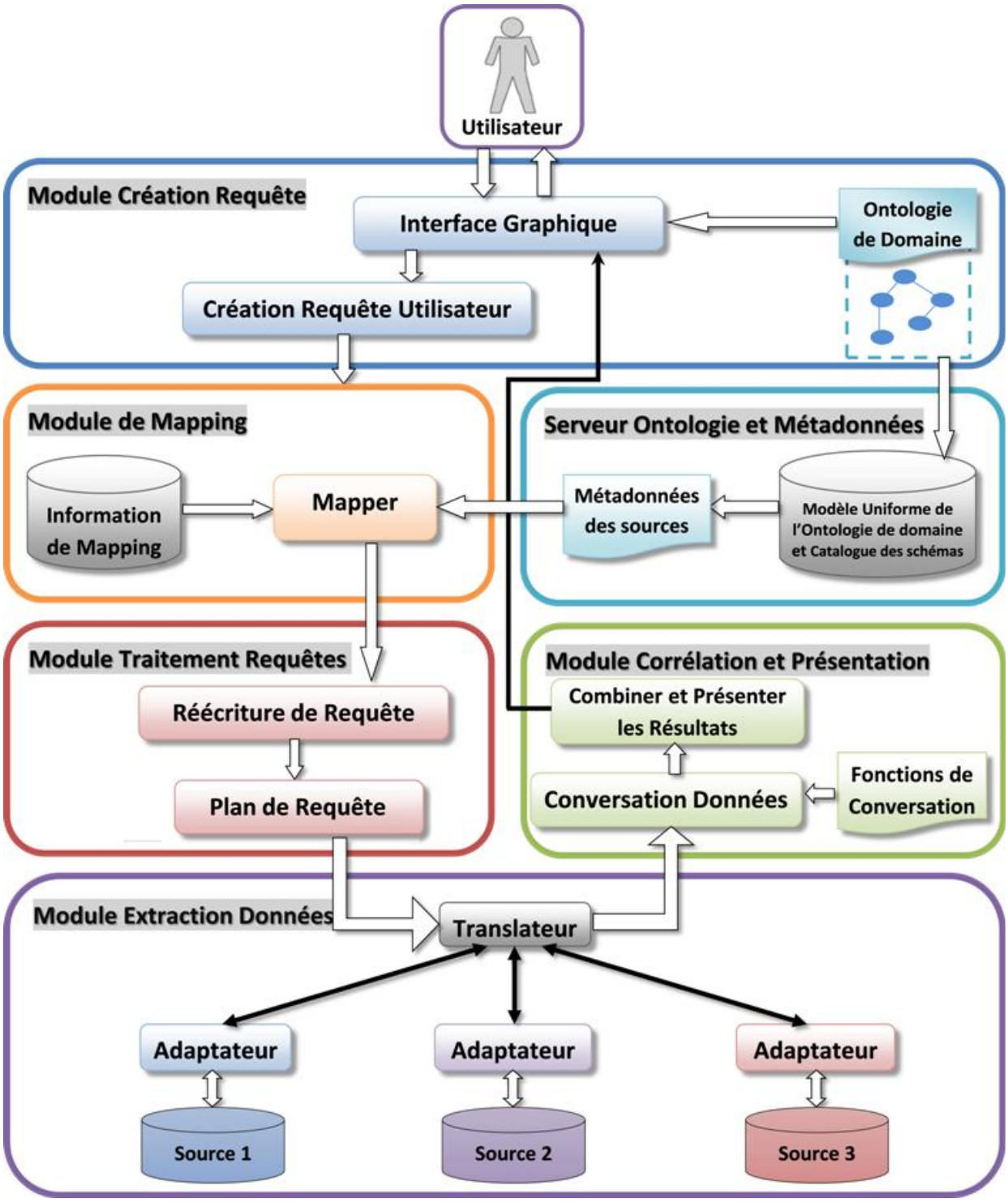


Fig. 3.7 – Architecture de système.

- ***Couche Présentation :*** Se compose de deux modules, un module de création de requête et un module de présentation et corrélation des données.
- ***Couche Application :*** Se compose de trois modules, un serveur d'ontologie et métadonnées des sources, un module de mapping sémantique et un module de traitement de requêtes.
- ***Couche Physique et données* :** Se compose d'un seul module, celui d'extraction de données.

3.1 Module de création des requêtes

Ce module se compose de deux tâches comme on le verra ci-dessous. Ces tâches impliquent directement les utilisateurs. L'interface graphique de notre système a été conçue pour faciliter l'interaction avec l'utilisateur.

3.1.1 Sélection de l'ontologie de domaine

A ce stade, une ontologie de domaine correspondant au domaine de la requête de l'utilisateur est sélectionnée. Cette ontologie de domaine spécifique contient tous les termes nécessaires pour exprimer la requête. Ensuite l'utilisateur examine l'ontologie et sélectionne les termes qui correspondent à la signification des termes de sa requête. Il est important que l'ontologie de domaine soit précise et complète pour le domaine concerné. Par exemple, considérons la requête de l'utilisateur suivante : *"Les Noms et les Emails de Staff Académiques dans l'Université".* Le système reçoit la requête et choisit l'ontologie de l'université comme l'ontologie de domaine spécifique. Nous supposons que la partie de l'ontologie montrée dans la Figure 3.2 est une partie de l'ontologie de l'université.

L'utilisateur ensuite, navigue dans cette ontologie, d'abord il choisit le concept père « University », puis le concept enfant « Staff » puis le sous-concept « AcadimicSatff » de cette ontologie qui correspond avec les concepts de Staff Académique de l'Université dans sa requête.

3.1.2 Création de requête

Après avoir sélectionné l'ontologie de domaine et les termes de l'ontologie de domaine correspondant à la requête, le module de création de requête construit une requête basée sur le langage de requête du système selon la syntaxe SQL comme suite :

SELECT *Name, Email* } *attributs de la requête*

FROM *AcadimicStaff* } *concept de la requête*

3.2 Serveur d'ontologie et métadonnées des sources

Ce module comprend les tâches suivantes :

3.2.1 Informations sur l'ontologie de domaine

Ce module fournit des informations sur les ontologies de domaine, Ces dernières sont tout d'abord transformées dans le modèle de représentation uniforme interne (voir section 2.3) avant qu'elles soient stockées dans le serveur d'ontologie.

3.2.2 Informations sur les sources de données (métadonnées)

Ce module fournit des informations sur les sources de données locales qui résident dans le système et que l'on veut intégrer. Le serveur métadonnées maintient un catalogue de sources, qui contient des informations sur ces sources de données.

Le serveur ontologie et métadonnées peut être invoqué par d'autres modules du système afin d'obtenir des informations sur la structure des sources de données locales. Il peut être utilisé pour les services suivants :

- Obtenir les termes de l'ontologie de domaine pour la création de requête utilisateur.
- Obtenir les termes existants dans les différentes sources de données locales.
- Obtention du mapping et les correspondances entre les termes des sources locales vers l'ontologie de domaine.

3.2.3 Choix de source de données locale

Une fois le module de création de requêtes a créé la requête de l'utilisateur, le serveur ontologie et métadonnées est invoqué afin de sélectionner les sources de données locales qui sont pertinentes au domaine de la requête de l'utilisateur. La tâche principale du serveur ontologie et métadonnées consiste à sélectionner les sources de données locales qui correspondent au domaine de la requête de l'utilisateur.

3.3 Module de mapping sémantique

Ce module a pour but d'effectuer un mapping sémantique entre les termes de la requête de l'utilisateur (choisi parmi les concepts de l'ontologie de domaine spécifique) et ceux dans les sources de données locales. Les termes de la requête de l'utilisateur sont remplacés par les termes des sources de données locales. A cet effet, l'ontologie de domaine choisie est transformée au modèle de représentation uniforme interne au système (voir la section 2.3). Ensuite, le module de mapping sémantique trouve les termes correspondants sémantiquement et remplace les termes de la requête avec ceux qui y correspondent dans les sources locales. Ainsi, ce remplacement, résoudra les conflits d'hétérogénéité sémantiques entre les termes de la requête de l'utilisateur et les sources de données.

Il existe trois principaux enjeux de mapping sémantique :

- Comment trouver les similarités sémantiques entre l'ontologie de domaine et les sources de données locales ?
- Comment représenter et décrire les informations de mapping sémantique ?
- Comment exécuter le mapping ?

En général, dans le processus de mapping sémantique, les relations de similarité entre les termes doivent d'abord être détectées et extraites, puis, représentées et stockées dans le schéma de mapping décrit dans la section 6. Ces relations de similarité (appelées informations de mapping sémantique) sont utilisées pour effectuer le mapping sémantique entre les termes, et que nous illustrons à travers l'exemple qui suit:

Les tâches générales pour le mapping sémantique entre l'ontologie de domaine et la source de données 1 (représentées dans les Figures 3.2 et 3.3) peuvent être les suivantes :

1. Extraction des concepts similaires : «People» avec «Staff», «Teacher» avec «AcadimicStaff », « Staff » avec « AdminStaff », sont des termes similaires les uns aux autres.
2. Mesure et détermination du type de relations de similarité entre les termes : Chaque approche peut envisager différents types de relations de similarité entre les termes (comme «Equivalentes», «Moins-générale», «Plus-générale» ou de «chevauchement», par exemple :
   (« GraduatProgram » a une relation moins générale avec « Program » et « Professor » a une relation de chevauchement avec « Lecturer »).
3. Représentation des relations de similarités entre les termes : dans cette étape, les similarités entre les termes sont formalisées. Par exemple, nous devrions représenter la relation de similitude entre « Teacher » avec « AcademicStaff » par un langage formel. Ces descriptions formelles de relations de similarité sont appelées informations de mapping sémantique.
4. Exécution de mapping sémantique entre les concepts similaires : dans cette étape, les concepts similaires les uns aux autres, sont mis en correspondance. Par exemple, « Teacher » est associé à « AcademicStaff ».

Le mapping sémantique est un processus difficile et complexe qui nécessite la détection et la mesure des similarités en termes, afin de représenter les informations de mapping dans un formalisme et exécuter le mapping sémantique.

3.4 Module de traitement des requêtes

Ce module comprend les tâches suivantes :

3.4.1 La reformulation de la requête de l'utilisateur

Après le processus de mapping sémantique des termes de sources locales à l'ontologie de domaine, la requête de l'utilisateur qui a été créée en termes de concepts de l'ontologie de domaine est reformulée en un ou plusieurs sous-requêtes. Le module de traitement de requête doit d'abord reformuler la requête en sous-requêtes qui se réfèrent directement à des schémas de sources de données locales. Pour ce faire, il doit disposer d'un ensemble de descriptions des sources de données locales. La description des sources de données spécifie le contenu des tables, des colonnes et des contraintes sur le contenu. Cette information existe dans le serveur d'ontologie et métadonnées. Par exemple si un utilisateur exprime une requête sur les caractéristiques d'un produit dont certaines de ses spécifications existent dans la source de données **A** et certaines dans la source de données **B**, le module de traitement de requête reformule la requête de l'utilisateur en deux sous-requêtes **A** et **B**.

3.4.2 Création du plan de requête

Après avoir sélectionné un ensemble minimal de sources de données pour une requête donnée, et que cette requête a été reformulée en termes des sources, un problème clé consiste à trouver le plan d'exécution des requêtes reformulées. En particulier, le plan de requête spécifiant l'ordre d'exécution des requêtes reformulées et l'ordre dans lequel s'effectuent les différentes opérations (jointure, sélection et projection).

3.5 Module d'extraction des données

Ce module comprend les tâches suivantes :

3.5.1 Traduction de requête de l'utilisateur au langage de requête de l'adaptateur

Une fois la requête reformulée en une ou plusieurs sous-requêtes et le plan de requête crée par le module de traitement des requêtes, chaque sous-requête est traduite au langage de requête de l'adaptateur correspondant.

3.5.2 Extraction des données

Les données sont extraites depuis les sources de données présélectionnées. Les adaptateurs sont utilisés pour extraire des données à partir des sources de données locales. Ce sont des modules qui comprennent l'organisation de données spécifique [26] [56]. Ils savent comment récupérer des données à partir de la source sous-jacente et cacher la structure des données spécifiques au reste du système d'information.

3.6 Module de corrélation des données

Ce module comprend les tâches suivantes :

3.6.1 Conversion des valeurs de données

Les données extraites obtenues depuis les adaptateurs sont envoyées au convertisseur. Ce dernier résout les conflits d'hétérogénéité de valeur de données entre les données extraites. Il exploite les fonctions de conversion et les règles de résolution des hétérogénéités entre deux valeurs de données (telles que les unités, types de données, des fonctions de conversion de format de valeur).

3.6.2 Corrélation des réponses

Après l'extraction et la conversion des réponses, celles-ci sont corrélées et combinées.

3.6.3 Présentation du résultat final

Ce module présente les données extraites sous un format de données commun.

4. Création de requête utilisateur

Notre approche proposée se base sur l'utilisation d'une ontologie de domaine. L'utilisateur se limite ainsi à utiliser une seule ontologie de domaine pour exprimer sa requête. Cependant, dans le contexte de notre travail, nous nous limitons à l'expression de requêtes simple. Les utilisateurs ne peuvent pas poser des requêtes complexes. La construction de la requête est basée sur une structure de requête interne au système et fondée sur des éléments de modèle de représentation uniforme interne (au système) de l'ontologie de domaine. Nous définissons la structure et la syntaxe SQL des requêtes comme suit :

SELECT < noms d'attributs >

FROM < nom de concept >

WHERE {< noms d'attribut OP valeurs> FROM <concept name1 >

< noms d'attribut OP valeurs> FROM <concept name2 >…} ;

L'utilisateur peut, interroger les attributs d'un seul concept de l'ontologie de domaine (que nous appelons le concept de la requête). L'utilisateur peut spécifier des contraintes et des conditions sur sa requête, exprimées dans la clause "WHERE". L'Opérateur (OP) peut être l'un des opérateurs SQL (=,<>,>,<,>=,<=, Between, Like or IN). Nous clarifions la syntaxe de la requête et sa structure avec l'exemple suivant :

Supposons qu'un utilisateur ait besoin des Noms et des Emails des professeurs de sexe féminin, dont l'âge est supérieur à 50 ans et de spécialité droit.

Cette requête est dans le domaine de l'université, donc nous supposons qu'il existe une ontologie de domaine de l'université dans le système. Ainsi, l'utilisateur parcourt les termes de l'ontologie de domaine d'université et choisit les termes de la requête, puis il construit une expression sur la base de la requête de l'utilisateur comme suit :

SELECT Name, Email } *attributs de la requête*
FROM Professor } *concept de la requête*
WHERE

Field = Law FROM Professor
Sex = Female FROM Professor
Age > 50 FROM Professor;
} *Contraints sur les concepts et attributs*

Il est à noter que l'utilisateur ne peut pas poser des requêtes complexes. Cependant, il peut décomposer sa requête en sous-requêtes simples et ensuite les soumettre au système; par exemple, si un utilisateur a une requête sur des attributs liés à deux concepts, il devra présenter deux requêtes distinctes pour le système. L'utilisateur peut aussi simplement spécifier des conditions pour les attributs de concepts qui sont dans le chemin de la requête. Le chemin à travers lequel un utilisateur parcourt la hiérarchie de l'ontologie de domaine pour atteindre les termes de la requête est appelé le chemin de la requête. Par exemple, dans la requête ci-dessus, les conditions qui ont été définies pour les attributs de *nom*, le *sexe* et *l'âge* sont tous liés à des concepts dans le chemin requête.

5. Les différents cas de mapping

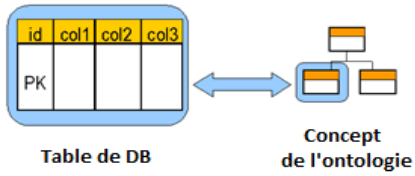
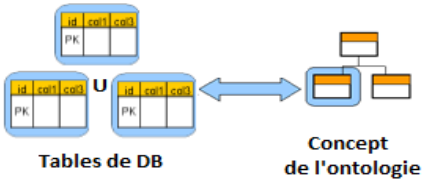
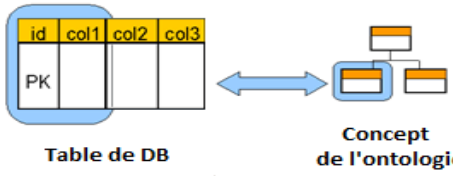
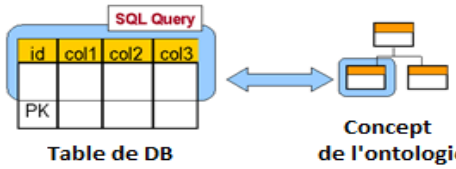
Après la création de la requête utilisateur depuis les termes de l'ontologie de domaine, le système doit trouver les correspondances entre ces termes et ceux des sources de données. Cette section présente les différents cas de mapping entre les schémas de bases de données relationnelles et l'ontologie de domaine qui sont prévus être couverts par notre approche. Le mapping peut être défini comme l'ensemble de correspondances qui lie le vocabulaire (les termes) d'un schéma local avec celui de l'ontologie de domaine. Autrement dit, nous voulons lier les tables, colonnes, et relations d'un schéma local avec les concepts, attributs et relations d'une ontologie de domaine.

A cause des critères de modélisation utilisés pour la construction des bases de données relationnelles qui sont différents de ceux utilisés pour la création des ontologies, les correspondances entre leurs éléments seront parfois délicats. Nous considérons les cas de mapping présentés dans le Tableau 3.1.

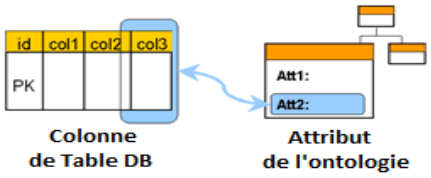
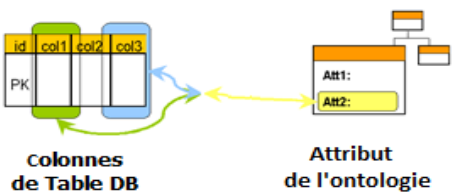
Cependant, Les valeurs des attributs et des relations peuvent être remplis directement à partir des valeurs des champs d'un enregistrement d'une base de données ou après application d'une fonction de transformation. Cette fonction peut affecter bien plus qu'un champ de données. Le Tableau 3.2 présente ces différents cas de mapping.

Bien que les opérations de l'algèbre relationnelle SQL couvrent de nombreux cas de transformations, il existe des situations dans lesquelles des transformations supplémentaires pourraient être nécessaires. Par exemple les opérations plus complexes comme les techniques de traitement du langage naturel sur les champs de données de texte, expressions régulières correspondant aux dates, l'extraction des URL ou Email, etc. Dans notre approche nous implémentons des convertisseurs. Ces derniers exploitent des fonctions de conversion et des règles de transformations entre deux valeurs de données (telles que les unités, types, format de valeur).

Tab. 3.1 – Classification des cas de mapping pour les concepts.

Le cas de mapping pour les concepts	Illustration graphique
Cas 1- Direct Mapping : Une table de DB correspond directement un concept dans l'ontologie. Chaque enregistrement de la table correspond à une instance d'un concept de l'ontologie.	
Cas 2- Union/Jointure : Un ensemble de tables DB correspond à un concept dans l'ontologie après jointure (Union). Chaque enregistrement résultant de jointure de ces tables correspond à une instance d'un concept de l'ontologie.	
Cas 3- Projection : Apparaît quand un sous-ensemble de colonnes d'une table DB sont nécessaires pour le mapping vers un concept de l'ontologie.	
Cas 4- Sélection : Apparaît quand un sous-ensemble de lignes d'une table DB correspond à un concept dans l'ontologie.	

Tab. 3.2 – Classification des cas de mapping pour les Attributs.

Le cas de mapping pour les attributs	Illustration graphique
Cas 1- Direct Mapping : Une colonne d'une table de DB correspond directement à un attribut d'un concept dans l'ontologie.	
Cas 2- Transformation : Une colonne d'une table de DB correspond à un attribut d'un concept dans l'ontologie après l'application d'une fonction de transformation.	
Cas 3- Concaténation : Apparaît quand la concaténation d'un sous-ensemble de colonnes d'une table DB est nécessaire pour correspondre à un attribut d'un concept dans l'ontologie.	

6. Schéma de mapping

Nous proposons un mécanisme de mapping entre les métadonnées des schémas locaux et le modèle de représentation uniforme de l'ontologie de domaine. Notre approche consiste à faire une correspondance entre ces structures et de stocker les informations de mapping dans un schéma méta-niveau relationnel appelé « Schéma de mapping ». Cette approche nous permet d'utiliser les technologies de bases de données relationnelles comme solution pour stocker les informations de mapping. Dans le cas le plus simple, une classe ou concept d'ontologie correspond à une table de base de données relationnelle, un attribut d'ontologie correspond à une colonne de la table, et une relation d'ontologie correspond à une clé étrangère. Dans les exemples de la vie réelle les correspondances ne sont pas si simples. Par exemple, un concept *Person* de l'ontologie pourrait être un domaine pour l'attribut *personAddress* de l'ontologie. Mais les tables de bases de données correspondantes au *Person* pourraient disposer d'une référence de clé étrangère vers une autre table ayant des informations d'adresse. Pour compliquer encore les choses, une propriété de type *xsd : string* peut correspondre à une combinaison de colonnes réparties sur plusieurs tables dans la base de données (par exemple : *pays, ville, rue* sont des informations stockées dans des tables séparées). Autres causes possibles de l'impossibilité de mapping direct sont les relations de sous-classe dans l'ontologie, l'utilisation de la cardinalité plusieurs à plusieurs, la non-existence de clés étrangères «naturel» dans les bases de données relationnelle. Souvent, les bases de données sont normalisées et leur structure est optimisée afin de maitriser les problèmes de la performance en cachant ainsi le vrai sens conceptuel. Pour faire face à ces contraintes, nous introduisons le schéma de mapping illustré dans la Figure 3.8.

Le modèle de représentation de l'ontologie de domaine est stocké dans les tables : *Ontologie, Concepts, Relations et Attributs*. Les métadonnées des sources de données sont stockées dans des tables : *Sources, Tables et Columns*. Pour chaque concept de l'ontologie la table *Concept_map* définit les connexions à des objets de base de données relationnelle qui peuvent être une table ou une vue (vue peut être définie en utilisant la colonne *Select_sql* de la table V*iews*). Dans les cas typiques, seules les vraies tables de base de données des sources existantes sont référencées (voir cas 1 Tab 3.1). Cependant, la technique des vues est utilisée pour gérer certaines situations de mapping délicat (voir cas 2 et 3 Tab 3.1). Chaque ligne de la table *Concept_map* contient une possibilité de filtrage sur l'objet de base de données relationnelles référencé (la colonne de *Filter_expr*) (voir cas 4 Tab 3.1).

La table *Attribut_map* détient les spécifications des instances des attributs de l'ontologie. Chaque enregistrement dans cette table se base sur un enregistrement *Concept_map* pour le domaine de l'attribut. Dans le cas le plus simple, il est évalué dans la table spécifiée pour le domaine *Concept_map*.

Notre schéma de mapping peut contenir des informations de mapping de plus d'une source à une ontologie de domaine. Pour chacune de ces sources, il devrait y avoir une ligne distincte dans la table *Sources* et qui sera une clé étrangère pour les tables : *Tables et Columns*. Comme pour le modèle de représentation de l'ontologie et les métadonnées des sources, nous utilisons une structure relationnelle pour stocker notre schéma de mapping en utilisant n'importe quel SGBD tel que MySQL.

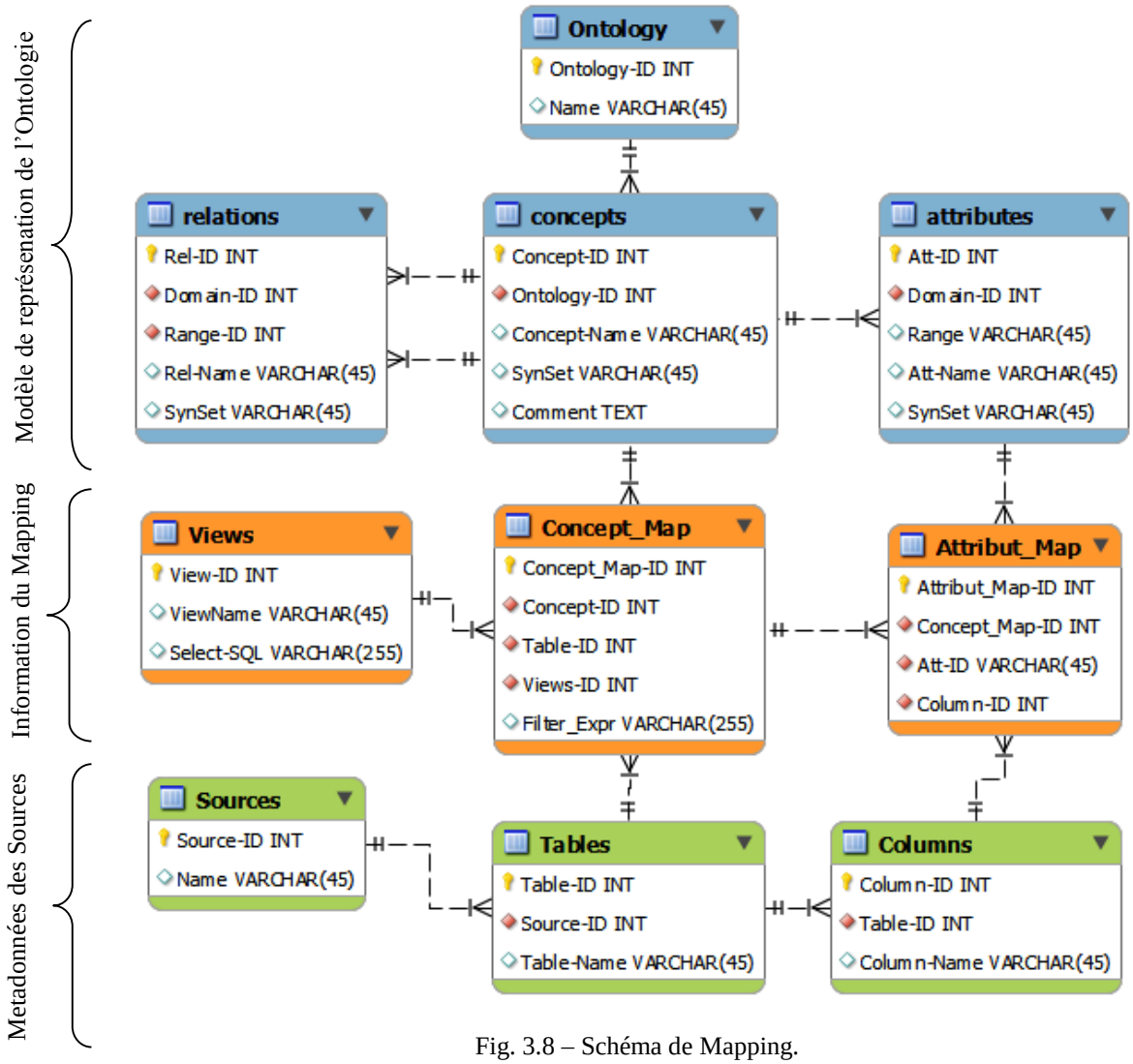


Fig. 3.8 – Schéma de Mapping.

6.1 Exemple de mapping

Pour mieux expliquer notre schéma de mapping, nous allons voir un exemple simple. Les Figures 3.10 et 3.9 représentent une partie de l'ontologie de domaine de l'université et un schéma de base de données locale respectivement. Par exemple, les classes *'Student', 'Course' et 'Program'* dans cette partie de l'ontologie de domaine ont les tables correspondantes :*"student", "course*" et *"program"* dans le schéma de base de données locale. De même, pour les classes 'AcademicStaff' et 'AdminStaff' de l'ontologie et dont leurs instances sont peuplées depuis les tables correspondantes : "teacher" et "staff" dans le schéma de base de données respectivement.

Les classes 'Lecturer', 'Researcher' et 'Professor' extraient leurs instances de données depuis une seule table "teacher" mais chacune utilise un filtrage différent. Cela signifie que la valeur de la chaine de caractères *'level_code =…'* doit être écrite dans *concept_map.filter_expr* dans la ligne correspondante.

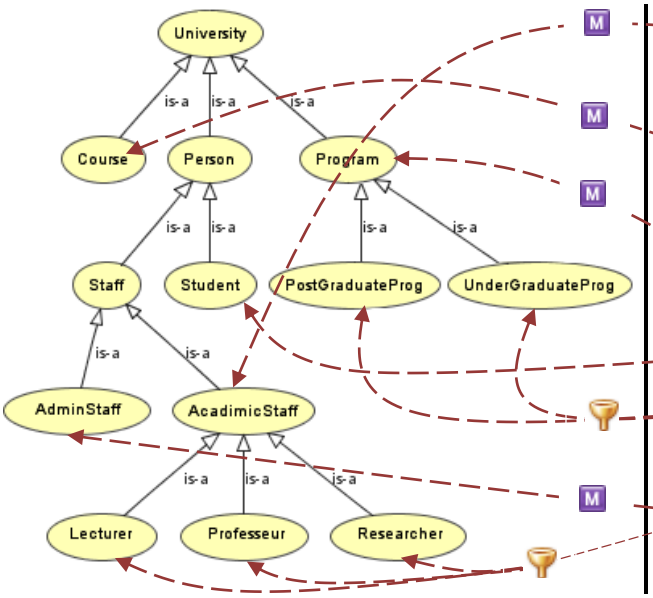


Fig. 3.10 – Une partie de l'ontologie de domaine Université.

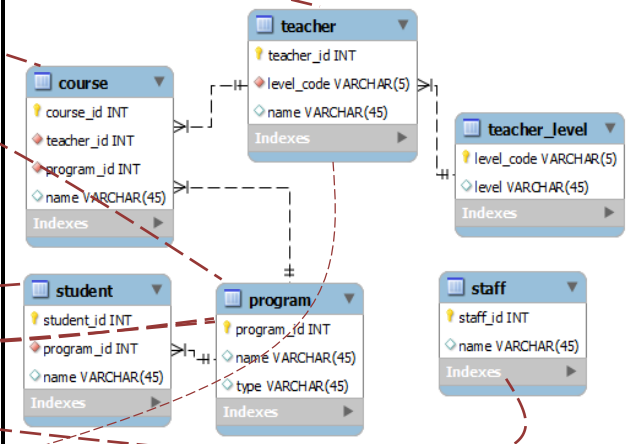


Fig. 3.9 – Exemple de source de donnée local.

De même pour les classes 'PostGraduateProg' et 'UnderGraduateProg', qui obtiennent leurs instances de données depuis la table "program" via un filtrage différent. Cela signifie que la valeur de la chaine de caractère *'prog_type =…'* doit être écrite dans *concept_map.filter_expr* dans la ligne correspondante. Le Tableau 3.3 présente le détail de la table *concept_map* de l'exemple.

Les instances de la classe 'Person' sont peuplées avec une vue d'union des tables "student","teacher" et "staff". Cette vue est stockée dans le champ *select_sql* de la table *Views* dans le schéma de mapping.

De même pour la class 'Staff', ses instances sont peuplées avec une vue d'union des tables "teacher" et "staff". Tableau 3.4 présente le détail de la table *Views* de l'exemple.

De même que les classes, les attributs de l'ontologie seront liés aux attributs du schéma de source de données locale, les informations du mapping sont stockées dans la table *attribut_map* du schéma de mapping. Le Tableau 3.5 présente le détail de la table *attribut_map* de l'exemple.

Si nous supposons par exemple que la requête de l'utilisateur créée par le système est :

SELECT Professor ID, ProfessorName
FROM Professor;

La reformulation de la requête pour la source correspondant en utilisant le schéma de mappage sera :

SELECT teacher_id, name
FROM teacher
WHERE
{ level_code='PROF'};

Tab. 3.3 – La Table *concept_map* de d'exemple.

Concept_Map-ID	**Concept-ID**	**Concept-Name**	**Table-ID**	**Table-Name**	**Views-ID**	**Filter_Expr**
1	52	Student	4	student		
2	19	Course	3	course		
3	25	Program	1	program		
4	42	AcademicStaff	5	teacher		
5	47	AdminStaff	6	staff		
6	43	Lecturer	5	teacher		level_code='LECR'
7	45	Professor	5	teacher		level_code='PROF'
8	46	Researcher	5	teacher		level_code='RESR'
9	27	GraduateProgram	1	program		type='Graduate'
10	30	UnderGraduateProgram	1	program		type='UnderGraduate'
11	61	Person			1	
12	41	Staff			2	

Tab. 3.4 – La Table *Views* de d'exemple.

View-ID	**ViewName**	**Select-SQL**
1	Person	*SELECT teacher_id, name FROM teacher **UNION** SELECT student_id, name FROM student **UNION** SELECT staff_id, name FROM staff*
2	Staff	*SELECT teacher_id, name FROM teacher **UNION** SELECT staff_id, name FROM staff;*

Tab. 3.5 – La Table *attribut_map* de d'exemple.

Attribut_ Map-ID	**Concept_Map-ID**	**Concept-Name**	**Att-ID**	**Att-Name**	**Column-ID**	**Column-Name**
1	1	Student	34	StudentID	10	student_id
2	1	Student	35	StudentName	12	name
3	2	Course	7	CourseTitle	9	name
4	6	Lecturer	15	LecturerID	13	teacher_id
5	6	Lecturer	16	LecturerName	15	name
6	7	Professor	20	ProfessorID	13	teacher_id
7	7	Professor	21	ProfessorName	15	name
8	8	Researcher	30	ResearcherID	13	teacher_id
9	8	Researcher	32	ResearcherName	15	name

7. Discussion

L'utilisation d'ontologies formelles dans notre approche permet une intégration automatique des sources données et un mapping semi-automatique à l'opposé des ontologies linguistiques où l'intégration est partielle (semi-automatique) ou même manuelle. Les ontologies formelles constituent des spécifications explicites de conceptualisations de domaines. Elles permettent de définir formellement les concepts ainsi que les relations inter-concepts du domaine étudié.

L'utilisation de métadonnées au niveau des sources de données et d'une ontologie globale dite de domaine au niveau du médiateur rend facile l'interopérabilité entre les données issues de sources distinctes et facilite la résolution des conflits structurels et sémantiques entre ces dernières.

Effectivement, les ontologies permettent l'intégration de sources de données (résolution des conflits sémantiques et schématiques), l'utilisateur dialogue avec le système en utilisant le vocabulaire de l'ontologie. Ce dernier le guide dans la formulation (conflits syntaxiques) et la création de ses requêtes. Notre approche présente les avantages suivants :

- Une intégration de sources de données relationnelles : des bases de données enrichies par des métadonnées.
- Le processus d'intégration est automatique via des ontologies formelles.
- L'utilisation de l'approche LaV (Local-as-View) : assure le passage à l'échelle et facilitant l'ajout des nouvelles sources de données.

Cependant elle présente les insuffisances suivantes :

- La complexité de réécriture et reformulation des requêtes.
- Les requêtes utilisateurs sont simples et se limitent au filtrage sur un concept (non prise en compte des jointures).
- La non prise en charge de cas mapping complexes tels que par exemple lorsqu' un concept de l'ontologie de domaine correspond à une colonne d'une table dans une source de données (ou le contraire) n'est pas encore couvert par notre système d'intégration.

8. Conclusion

Dans ce chapitre, nous avons proposé, une approche et une architecture de système pour l'intégration des sources de données relationnelles structurées. Cette approche utilise des ontologies pour résoudre les conflits sémantiques entre les sources de données. La résolution des conflits sémantiques entre les termes de la requête de l'utilisateur et ceux des sources de données est le principal problème décrit dans ce chapitre et qui est résolu grâce à l'utilisation d'ontologies.

L'approche proposée utilise des ontologies spécifiques au domaine pour la création des requêtes utilisateurs. Il existe une ontologie de domaine spécifique pour chaque domaine d'application, qui couvre toute la définition sémantique des concepts existants dans ce domaine. Nous introduisons un modèle de représentation uniforme pour les ontologies et nous proposons une approche pour la création des requêtes des utilisateurs basée sur ce modèle de représentation. L'utilisateur peut parcourir l'ontologie de domaine et choisir des termes et des concepts pour sa requête, le système crée par la suite une requête basé sur le langage de requête du système de syntaxe SQL.

Pour faciliter le processus de mapping, nous stockons les métadonnées de toutes les sources de données dans le système afin de définir toutes les informations de mapping entre les termes de l'ontologie de domaine et les sources de données locales. Nous proposons un mécanisme de liaison entre les métadonnées des schémas locaux et le modèle de représentation uniforme de l'ontologie. Cette approche nous permet d'utiliser la technologie de base de données relationnelle comme solution pour le stockage puis l'extraction d'informations de mapping.

Les termes de la requête de l'utilisateur sont mappés vers les termes des sources de données locales et la requête de l'utilisateur est ensuite réécrite en utilisant les termes de la source de données locale. Enfin, elle est traduite au langage de requête des sources de données correspondant et les données sont extraites, converties et présentées au format commun à l'utilisateur.

Notre système d'intégration couvre les niveaux d'abstraction des conflits d'hétérogénéité entre les sources de données. Le système présente :

- Ontologie : comme une solution pour résoudre les hétérogénéités sémantiques des schémas.
- Adaptateur : comme solution pour résoudre les hétérogénéités syntaxiques des modèles.
- Convertisseur : comme solution pour résoudre les hétérogénéités de valeur de données.

Il s'adapte à n'importe quel domaine par l'ajout d'une ontologie de domaine. Cela signifie que pour pouvoir utiliser le système dans n'importe quel domaine d'application, nous devons développer et ajouter une ontologie de domaine au système. Dans le chapitre suivant nous présentons l'implémentation du Prototype (HERO-Med) de notre approche d'intégration dans le domaine de l'éducation supérieure et des universités.

Chapitre 4
Implémentation et mise en œuvre

« Connaître, ce n'est point démontrer, ni expliquer. C'est accéder à la vision. Mais pour voir, il convient d'abord de participer : cela est un dur apprentissage . . . »

« La vérité de demain se nourrit par l'erreur d'hier. »

Antoine de Saint-Exupéry

Chapitre IV

Implémentation et mise en œuvre

Sommaire

1 Introduction 99
2 Création de l'ontologie de domaine de l'université 100
2.1 Méthodologie de construction des ontologies de domaine 100
2.2 Développement de l'ontologie de domaine de l'université 102
3 Interface utilisateur graphique 107
3.1 Barre de Navigation à gauche 108
3.2 Le Header des Traitements en haut 108
3.3 Détails en milieux 108
4 Module Ontology integration presperctive 108
4.1 Implémentation des Translators 112
4.2 Implémentation des Adaptateurs 113
4.3 Implémentation de l'Exportateur 113
5 Module Sources de données hétérogènes 115
6 Module information de Mapping 117
6.1 Implémentation de module de Mapping 117
6.2 Implémentation des Filtres 119
6.3 Implémentation des Convertisseurs 119
7 Module Reporting 119
8 Conclusion 119

1. Introduction

Afin de valider notre proposition, nous l'avons prototypée. Dans ce chapitre, nous nous concentrons sur la mise en œuvre et l'implémentation du Prototype HERO-Med (Higher Education Reference Ontology Mediator). Notre domaine d'application cible est celui de l'éducation supérieure et les universités, Il s'agit de pouvoir intégrer automatiquement les sources de données hétérogènes de différentes universités à travers notre Prototype, en offrant, sans aucune programmation, des possibilités d'accès ergonomiques. Par exemple les étudiants peuvent poser des requêtes sur les programmes, les cours, les enseignants, le staff administratif et peuvent accéder à des informations sur les nouvelles publications, des événements...etc. Nous avons choisi la Plateforme Web pour implémenter notre Prototype. Une application Web est une application manipulable grâce à un navigateur Web. Les applications Web présentent plusieurs avantages surtout dans le contexte ou un nombre important d'utilisateurs vont accéder à cette application pour intégrer et partager des informations. La maintenance est réduite, aucune application à installer sur les postes des utilisateurs, et facilement adaptable à une architecture orientée service (SOA).

Dans la section 2, nous présentons une méthodologie pour la construction des ontologies de domaine, ensuite, nous créons une ontologie de domaine spécifique dans l'éducation supérieure et les universités et la représenter dans notre modèle de représentation uniforme interne. La section 3 présente brièvement l'environnement de développement adopté et la technologie utilisée, ainsi que l'environnement d'exécution du Prototype HERO-Med. Dans la section 4, nous développons un module graphique pour la présentation de l'ontologie de domaine et la création précise et exacte des requêtes utilisateurs. Le processus de création et reformulation des requêtes utilisateur est illustré par un exemple. Dans la section 5, nous appliquons le processus d'intégration sur deux exemples de sources de données relationnelles hétérogènes en utilisant le mapping sémantique. Ensuite, dans la section 6, nous utilisons ces sources de données locales dans un scénario d'intégration en exploitant le module de mapping sémantique. Enfin, dans la section 7, nous discutons la conclusion de ce chapitre.

2. Création de l'ontologie de domaine de l'université

Notre système d'intégration des sources données proposé est basé sur les requêtes et une ontologie de domaine spécifique. Cela signifie que la sortie du système est une réponse à une requête posée par l'utilisateur dans un domaine spécifique, en fonction des termes de l'ontologie de domaine. Afin d'étendre notre système à un autre domaine, il est nécessaire de créer une ontologie de domaine qui comporte tous ses termes et la sémantique. Nous avons besoin des ontologies de domaine pour mettre en œuvre avec succès notre approche d'intégration de données. La qualité des résultats de l'intégration a un rapport direct avec la complétude et l'exactitude de l'ontologie de domaine.

A cet effet, nous discutons d'abord la méthodologie pour construire des ontologies de domaine basées sur notre modèle de représentation uniforme interne.

2.1 Méthodologie de construction des ontologies de domaine

Plusieurs approches de création d'ontologies ont été proposées dans la littérature. [59] ; [66] ; [67]. Comme mentionné dans [67], il y'à quelques règles fondamentales en matière de conception des ontologies de domaine que nous devons noter :

- Il n'existe pas une bonne façon de modéliser un domaine – il y'a toujours des alternatives variables.
- Le développement de l'ontologie est nécessairement un processus itératif.
- Les concepts de l'ontologie doivent être le plus proche possible des objets (physiques ou logiques) et les relations dans le domaine d'intérêt.

Nous avons certainement besoin de réviser l'ontologie initiale. Ce processus de conception itératif continuera probablement à travers tout le cycle de vie de l'ontologie. Nous suggérons les étapes suivantes pour la création des ontologies représentées dans notre modèle interne proposé (voir Figure 4.1).

Étape 1. *Détermination du domaine de l'ontologie :* nous commençons le développement d'une ontologie en clarifiant son domaine par répondre à plusieurs questions de base [67] :

- Quel est le domaine que l'ontologie ?
- Pour quelle raison nous allons utiliser l'ontologie ?
- Pour quels types de questions les informations dans l'ontologie doivent apporter des réponses ?
- Qui va utiliser et maintenir l'ontologie ?

Nous devrons rester loin des exigences applicatives et essayer d'être aussi général que possible dû fait que les ontologies de domaine décrivent l'intention universelle des termes d'une communauté [67].

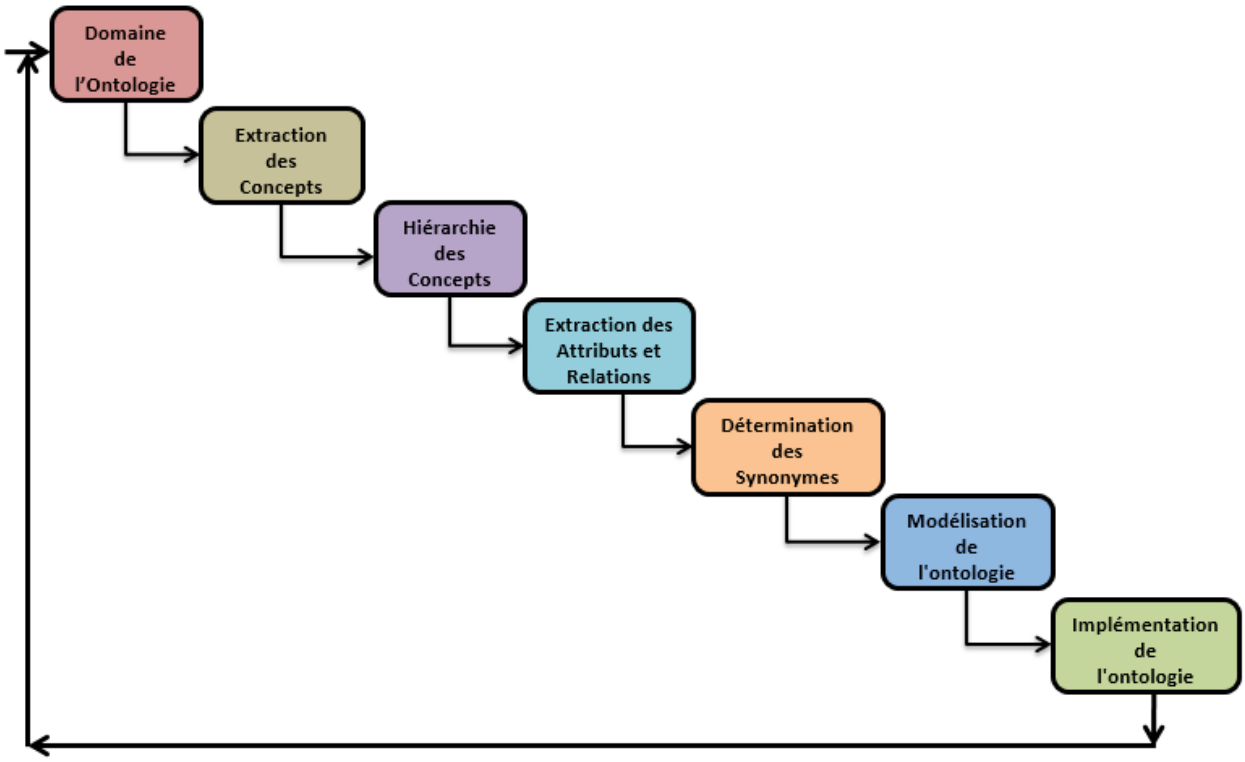


Fig. 4.1 – Étapes de création des ontologies de domaine.

Étape 2. *Extraction des Concepts (ou instances) :* Basée sur le domaine d'application défini, nous extrayons des concepts existants et nécessaires dans l'environnement de conceptualisation pour le domaine d'intérêt.

Étape 3. *La création de la hiérarchie des concepts :* dans cette étape nous organisons les concepts et les instances dans une hiérarchie taxonomique (super-concept, sous-concept). Si un concept A est un super-concept de concept B, alors chaque instance de B est aussi une instance de A. En d'autres termes, le concept B représente un concept qui est «*is-a*» A. Il n'ya pas une hiérarchie de concept unique et correcte pour un domaine donné. La hiérarchie dépend de l'utilisation possible de l'ontologie, le niveau de détail qui est nécessaire pour l'application, les préférences personnelles, et parfois les exigences de compatibilité avec d'autres modèles [67].

Étape 4. *Extraction des Attributs et des Relations :* Basée sur le champ d'application défini, nous extrayons les attributs pertinents de chaque concept et les relations existantes entre ceux extraits dans l'environnement de conceptualisation pour le domaine d'intérêt.

Étape 5. *Détermination des Synonymes :* nous spécifions les synonymes pour chaque concept, attribut et relation. Les synonymes de chaque terme est un autre nom qui est utilisé par d'autres communautés pour représenter le même terme.

Étape 6. *Modélisation :* nous modélisons les termes extraits dans un modèle de représentation des ontologies. Nous pouvons utiliser à cet effet, un outil de création des ontologies ou un éditeur d'ontologie comme Protégé 3.5, OntoStudio[16], ou Ontolingua[17].

[16] http://www.ontoprise.de
[17] http://ontolingua.stanford.edu

Étape 7. *Implémentation :* Enfin nous implémentons le modèle de représentation de l'ontologie de domaine dans un modèle de base de données ou un fichier.

Une question pratique souvent posée est : « q*uel est le profil de la personne qui doit développer les ontologies ?* » La personne qui se charge de créer et développer des ontologies doit avoir une bonne compréhension du vocabulaire et la conceptualisation du domaine. Cette connaissance permet d'assurer la conformité des ontologies avec la conceptualisation de la communauté en tant que mesure de la qualité pour les ontologies [117].

2.2 Développement de l'ontologie de domaine de l'université

Nous avons suivi la méthodologie présentée ci-dessus pour la construction de l'ontologie de domaine:

Étape 1. *Détermination du domaine de l'ontologie :* le domaine d'application de notre ontologie de domaine est l'éducation supérieure et l'université.

Étape 2. *Extraction des Concepts (ou instances) :* Pour l'extraction de concepts existants dans le domaine de l'éducation supérieure et l'université, nous avons besoin d'enquêter et d'étudier la conceptualisation de ce domaine en détails.

A cet effet, nous avons utilisé des sources d'information, des Sites Web des universités, des vocabulaires spécifiques universitaires et d'autres ontologies des universités (comme l'ontologie universitaire de SHOE « Simple HTML Ontology Extensions »).

Étape 3. *La création de la hiérarchie des concepts :* La Figure 4.2 présente la hiérarchie de concepts créée dans une forme taxonomique.

Étape 4. *Extraction des Attributs et les Relations :* Nous avons déterminé les attributs de chaque concept et les relations entre les concepts extraits.

Étape 5. *Détermination des Synonymes :* Nous avons utilisé des dictionnaires et des Sites Web des universités pour préciser les synonymes de chaque terme extrait (concepts, attributs et relations).

Enfin, nous avons abouti au développement de l'ontologie ci-dessous. Nous présentons d'abord la hiérarchie de concept créée dans une forme taxonomique. Nous présentons les attributs extraits pour chaque concept entre [crochets], et les synonymes entre {accolades}. Ensuite, les relations entre les concepts, et l'ensemble des synonymes pour les attributs et les relations.

Hiérarchie de concepts:

University *{University college} [home-page, name, country]*
- AboutUniversity *{Overview, About-us, About-college} [home-page]*
  - ContactInfo *{Contact, Phone directory}*
  - History *{University History}*
  - MapLocation *{University County, Situation, Localization}*
  - PresidentMessage *{Chair-message, Mission}*
- Degree *{Diploma}*
  - Associate *{Associate Degree, Diploma}*
  - Bachelor *{Baccalaureate Degree, Diploma}*
  - Doctorate *{Doctorate Degree, Diploma}*
  - Master *{Master Degree, Diploma}*
- Event *{News, Event-news} [home-page]*
  - Conference *{Symposium, discourse, Disquisition, huddle} [Web-page, Description, Event-date]*
  - News *{Event, News-event} [Web-page, Description, Event-date]*
  - Seminar *{Seminary, Colloquy} [Web-page, Description, Event-date]*
  - Workshop *{Demos show, Work studio, Shed} [Web-page, Description, Event-date]*
- Faculty *{Faculty, School, Academic-Department}*
  - Department *{Academic-Department} [home-page, name]*
    - Course *{Subject, Module} [Web-page, title]*
    - Program *{Degree-program, Academic-program, Educational-degree} [home-page] (key-relationship: list-of-course)*
      - UnderGraduateProgram *{First tertiary degree} [Web-page]*
        - BachelorProgram *{Baccalaureate-program} [Web-page, Field]*
        - DiplomaProgram *{Diploma-program} [Web-page, Field]*
      - PostGraduateProgram *{Graduate-program} [Web-page]*
        - DoctorateProgram *{PhD, Doctor-of-philosophy} [Web-page, Field]*
        - MasterProgram *{Master-of-science} [Web-page, Field]*
      - PostDoctoralProgram *{Postdoctoral-position, Postdoctoral-fellowship} [Web-page, Research-field]*
  - Laboratory *{Lab, Work-lap} [home-page, name]*
    - ResearchGroup *{Research-team, Research} [home-page, Research-field] (key-relationship: Research-field)*
- Publication *{Research-publication}[home-page]*
  - Article *{paper}*
    - ConferencePaper *{Conference-proceeding} [Web-page, Title, publish-date, Author] (key-attribute: author)*
    - JournalArticle *{Journal-publication, Journal-paper} [Web-page, Title, publish-date, Author] (key-attribute: author)*
  - Book *{Work, Quid, Paper, Chapter} [Web-page, Title, publish-date, Author] (key-attribute: author)*
  - TechnicalReport *{Scientific-report}*
  - Thesis *{Dissertation, Theory}*
    - DoctoralThesis *{PhD-thesis} [Web-page, Title, publish-date, Author] (key-attribute: author)*
    - MastersThesis *{Master-thesis} [Web-page, Title, publish-date, Author] (key-attribute: author)*
- Project *{Plan, Operation, Idea, Design} [Project-Title, home-page] (key-relationship: Project-Title)*
- Person *{Humain bieng, Individual, Self } [Name,Email, home-page]*
  - Staff *{employee, worker, people, educational-employee} [home-page]*
    - AcademicStaff *{Faculty, Academic-people, faculty-member, academician, academic} [home-page]*
      - Professor *{Prof} [URI] (key-relationship: teach)*
      - Lecturer *{Instructor, Trainer, Educator} (key-relationship: teach)*
      - Researcher *{Research-worker, Scientist} [name, Email, Research-field, home-page] (key-relationship: research-field)*
    - AdministrativeStaff *{Non-Academic-Staff, Management-staff}*
      - UniversityDean *{Chair} [name, Email, home-page]*
      - Dean *{dean- assistant}*
      - Head-Department *[name, Email, home-page, Department-name]*
      - AdministrativeAssistant *{administrator}*
      - AssistantRegistrar *[name, Email]*
      - OfficialAssistant *[name, Email]*
  - Student *{Current-student, Prospective student}[home-page] (key-relationship: studyin)*
    - UnderGraduateStudent
      - DiplomaStudent *[home-page]*
      - BachelorStudent *{degree student}[name, Email, home-page]*
    - PostGraduateStudent *{Graduate-Student}*
      - MasterStudent *{Master-Student} [name, Email, home-page]*
      - DoctorateStudent *{PhD-Student} [name, Email, home-page]*
    - PostDoctoralStudent *{Post-doc-student} [name, Email, Research-field, home-page]*

Fig. 4.2 – Hiérarchie des concepts de l'ontologie de domaine de l'Université.

Les Relations entre concepts:

Advisor *(AcademicStaff, Student)*
BelongsTo *(Researcher,ResearchGroup)*
ComposedOf *(ResearchGroup,Researcher)*
Contains *(Laboratory, ResearchGroup)*
EnrolledBy *(Studen, Universityt)*
HasDegree*(Faculty,Degree)*
HasDoctorate *(AcademicStaff,Doctorate)*
HasMaster *(Researcher, Master)*
ProjectPublication *(Project, Publication)*
HeadOf *(University, UniversityDean)*
HeadOf *(Faculty, Dean)*
HeadOf *(Department, HeadDepartment)*
HeadOf *(HeadLibrary, Library)*
WorkIn *(Staff, Department)*
WorkIn *(ResearchGroup, Project)*
Teach *(AcademicStaff, Subject)*
MemberOf *(Staff, Department)*
MemberOf *(Staff, Project)*
StudyIn *(Student, Program)*
PublishedBy *(Publication, Researcher)*
Offers *(Department, Program)*
ListOfCourse *(program, Subject)*

Les Synonymes des Attributs :

URL: *{URI, Web-address, URL, Home-page}*
Name: *{Full-name, first-name, last-name, surname, given-name}*
Email: *{electronic-mail, mail}*
Phone: *{telephone, hand-phone, phone-number, Contact-number}*
Homepage: *{personal-Web-page, personal-homepage, personal-page, URL}*
Title: *{subject}*
Field: *{area, major, program}*
ResearchField: *{Research-interest,Research-area}*
Address: *{Permanent-address, Postal-address, Add.}*
PublishDate: *{Pub-date, date, year}*
Author: *{writer, Published-by, Publisher}*
Position: *{Post, job, Job-title}*
EventDate**:** *{start-date, date, deadline,important-date}*
Description**:** *{Description}*
ProjectTitle**:** *{title, subject}*

Les Synonymes des Relations:

Advisor: *{supervisor}*
BelongsTo: *{appertain}*
ResearchPublication: *{Publication}*
EnrolledBy: *{Register,Sign on, Enter, Join}*
HeadOf: *{Manager, Chair}*
WorkIn: *{employ}*
Teach: *{educate}*
StudyIn: *{Learn}*
PublishedBy: *{Put out, Bring out}*
MemberOf: *{work, team}*
Offers: *{Program}*
ListOfCourse: *{list-of-subject, course-material, curricula}*

Étape 6. *Modélisation :* Dans cette étape, nous avons conçu un graphe de hiérarchie de concepts et un graphe de relations de concepts en utilisant l'éditeur d'ontologie Protégé 3.5 (voir Figure 4.3). Protégé 3.5 a été développé par le groupe de « *Mark Musen à Stanford Medical Informatics* ».

Une partie de codage OWL (notation RDF/XML) de l'ontologie de domaine Université est détaillée dans l'Annexe A.

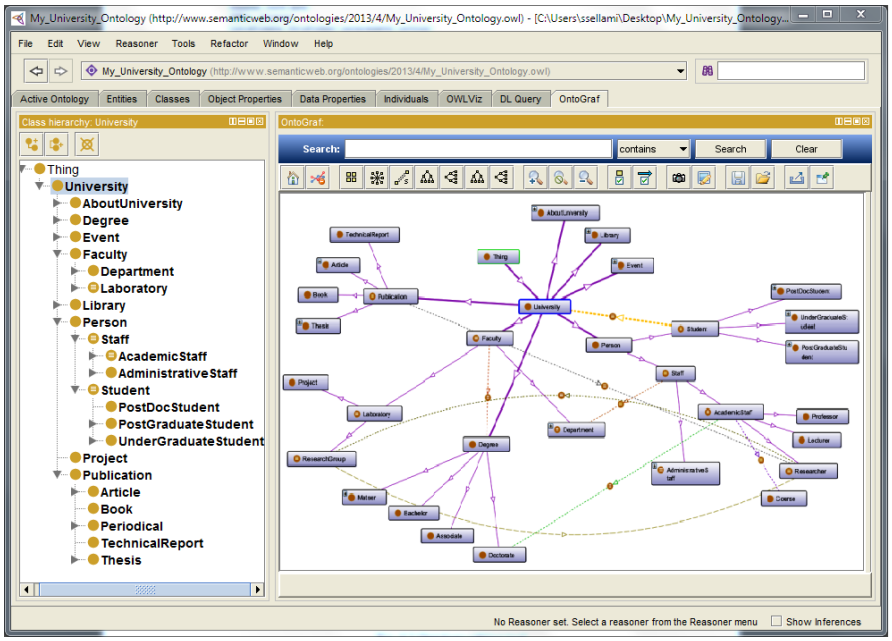


Fig. 4.3 – Editeur d'ontologie « Protégé ».

Étape 7. *Implémentation :* Enfin, nous avons implémenté l'ontologie ci-dessus après sa transformation au modèle de représentation uniforme interne proposé, et l'a stockée dans une base de donnée sous le SGBD MySQL 5.1 (voir Figure 4.4).

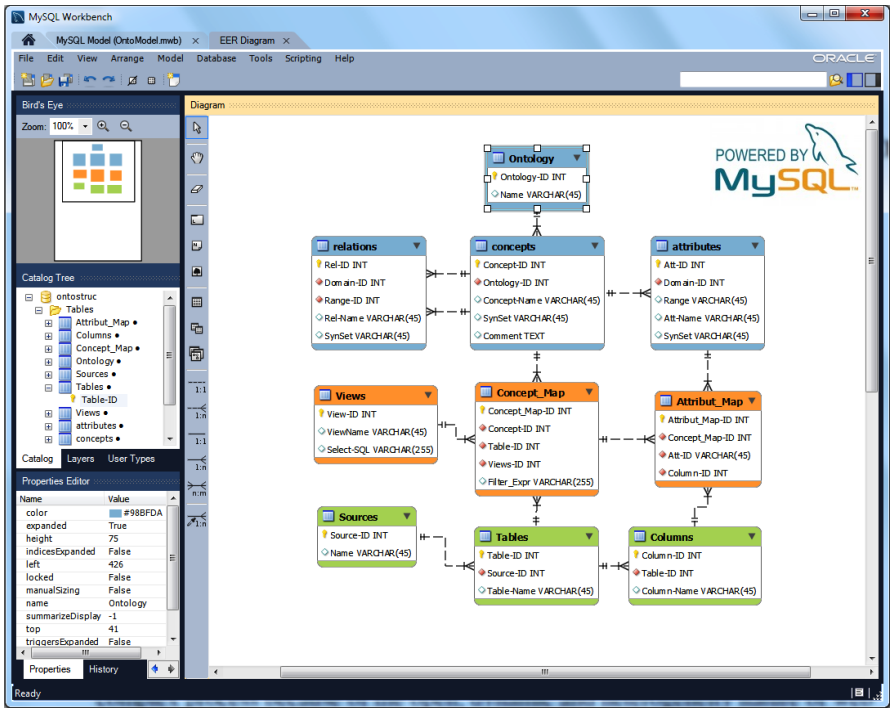


Fig. 4.4 – Le modèle de représentation uniforme de l'ontologie de l'Université.

3. Interface utilisateur graphique

Les utilisateurs interagissent avec le système via une interface graphique. Nous avons implémenté l'interface graphique (voir Figure 4.5), et les différents modules du Prototype HERO-Med en utilisant :

- **Comme Environnement de développement,** les langages de la Plateforme .NET : ASP.Net [18], C# et JavaScript dans l'environnement IDE Visual Studio 2012 [19].
- **Comme Environnement d'exécution,** le Serveur Web IIS 7 (Internet Information Services: Express 7) [20] pour exécuter le Prototype dans le navigateur Mozilla Firefox 26 [21] sur une machine Windows 7 avec une configuration matérielle acceptable (Processeur : Intel Core i5, RAM : 2Go).

Une partie du code de l'implémenation de notre interface graphique est présentée dans l'Annexe C. Notons que le Prototype HERO-Med est compatible avec n'importe quel navigateur web qui supporte ASP.Net et Java Script tel que: (Internet Explorer, Google Chrome, Opera, .. etc.).

L'interface principale du Prototype HERO-Med est divisée en trois parties :

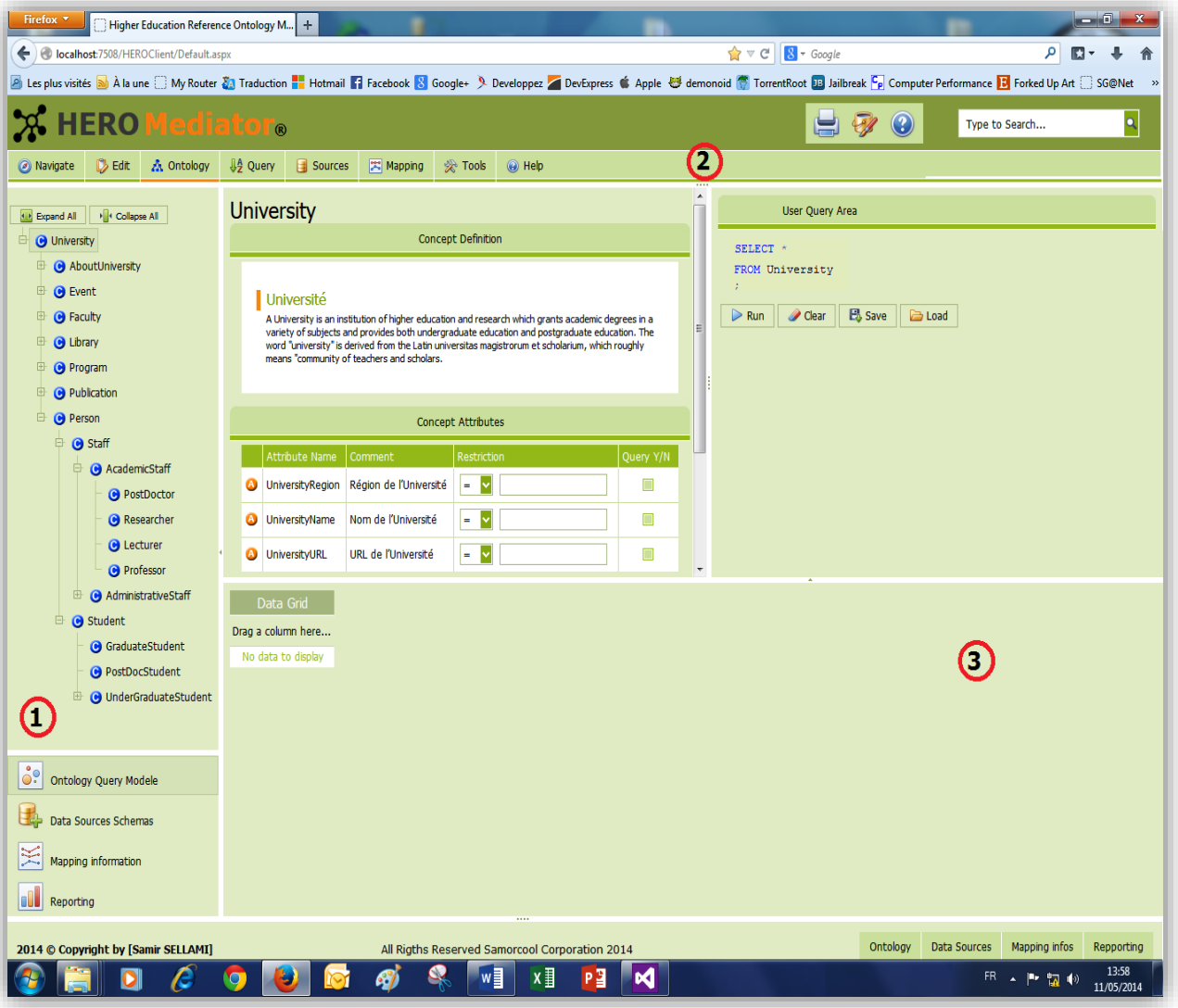


Fig. 4.5 – L'interface utilisateur graphique « GUI » du Prototype HERO-Med.

[18] http://www.asp.net
[19] http://www.visualstudio.com
[20] http://www.iis.net/
[21] http://www.mozilla.org/firefox/

3.1 Barre Barre de Navigation à gauche : Cette barre permet de naviguer entre les différents modules du Prototype HERO-Med : (Module Ontology integration perspective, Modules Sources de données, Module information de Mapping, Module Reporting).

3.2 Le Header des Traitements en haut : la partie Header contient le Menu des traitements des différents modules, ainsi que les opérations globales du Prototype HERO-Med telle que : (l'impression, changement du thème, l'accès à l'aide et la recherche par mot clé des concepts).

3.3 Détails en milieu : la partie détails affiche les pages et l'interface du module en cours d'exploitation.

4. Module Ontology Integration Perspective

L'interface graphique doit afficher la hiérarchie de l'ontologie de domaine pour l'utilisateur afin qu'il puisse trouver facilement et rapidement les termes de sa requête. Cette dernière sera créée automatiquement par le système.

Dans notre approche, la première interface graphique affiche une liste des ontologies de domaine. L'utilisateur choisit une ontologie de domaine lié à sa requête. Ensuite, l'interfance affiche une hiérarchie de concepts de l'ontologie de domaine à l'utilisateur (concepts dans les niveaux supérieurs de l'arbre de la hiérarchie de concepts). L'utilisateur choisit un des concepts. Ce dernier représente la racine de la requête. La Figure 4.6 montre l'interface correspondant au concept « Student » de l'ontologie de domaine Université. L'utilisateur peut avoir tout le détail de la hiérarchie de l'ontologie de domaine ou la réduire en utilisant les fonctionnalités « *Expand All* et *Collapse All* ».

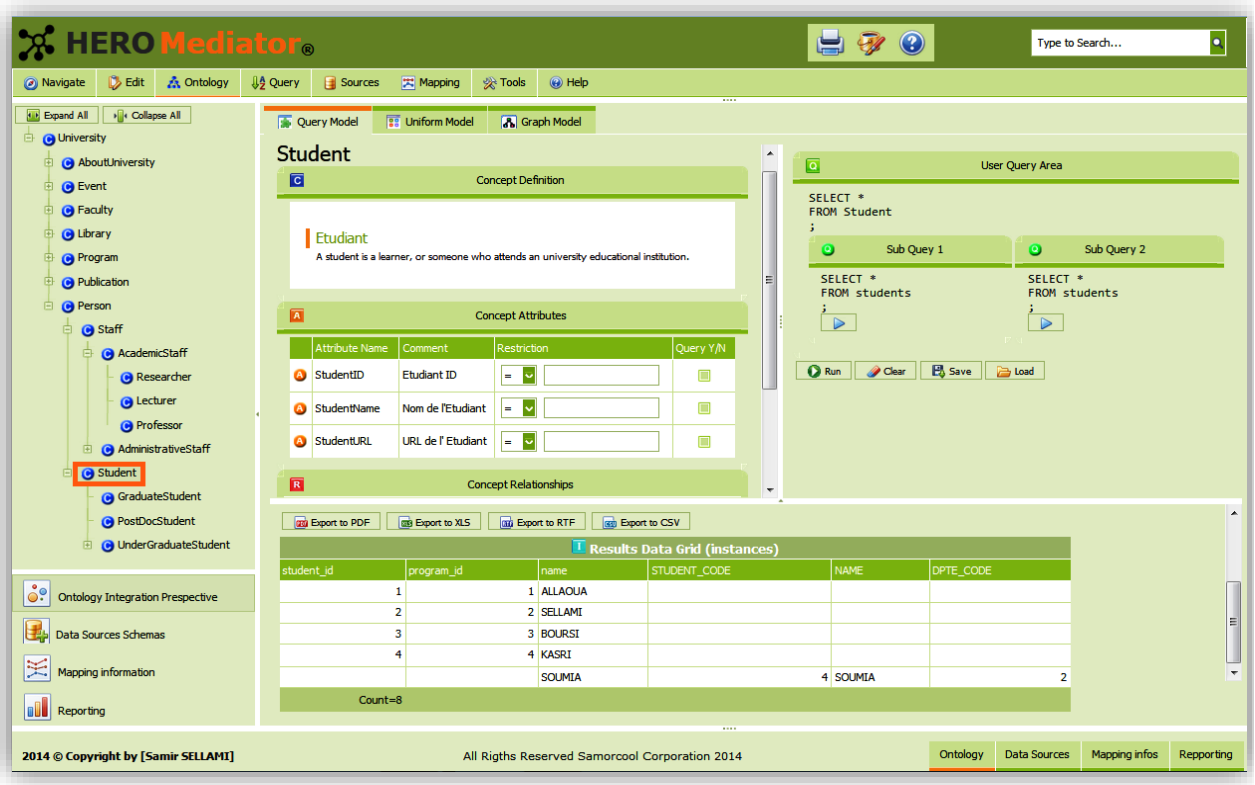


Fig. 4.6 – L'interface d'affichage de détail du concept « Student » de l'ontologie.

Notre approche proposée est une approche spécifique au domaine vu que l'utilisateur se limite au choix des termes d'une ontologie de domaine spécifique pour sa requête. Cette approche peut être étendue à tous les domaines à condition que l'ontologie de domaine pertinente soit déjà développée. L'utilisateur se limite à utiliser une seule ontologie de domaine pour sa requête. Les utilisateurs ne peuvent pas poser des requêtes complexes, vu que la construction de la requête est basée sur une structure de requête interne au système et fondée sur des éléments de modèle de représentation uniforme interne (au système) de l'ontologie. Nous définissons la structure et la syntaxe SQL illustrée ci-dessous en tant que structure de requête du système et la syntaxe de l'expression de la requête de l'utilisateur.

SELECT < noms d'attributs >

FROM < nom de concept >

WHERE { < noms d'attribut OP valeurs> FROM <concept-name >

L'utilisateur peut, dans cette structure de requête, interroger les attributs d'un seul concept de l'ontologie de domaine (que nous appelons le concept de la requête). Il peut spécifier des contraintes et des conditions sur sa requête, qui sont exprimées dans la clause "WHERE". L'Opérateur (OP) peut être l'un des opérateurs SQL *(=,<>,>,<,>=,<=, Between, Like or IN).* Nous clarifions la syntaxe de la requête et sa structure avec l'exemple suivant : *Supposons qu'un utilisateur a besoin des Noms et des Emails des professeurs en droit dans les universités qui ont plus de 50 ans et sont de Sexe féminin*.

L'utilisateur parcourt l'ontologie de domaine et choisit les termes de la requête. Le système construit ensuite l'expression suivante:

SELECT Name, Email } *attributs de la requête*
FROM Professor } *concept de la requête*
WHERE

Field = Law and
Sex = Female and
Age > 50 ;
} *Contraints sur les attributs*

Il est à noter que du fait que les requêtes doivent être simples, l'utilisateur peut décomposer sa requête complexe en sous-requêtes simples et ensuite les soumettre au système. Par exemple, si la requête porte sur des attributs liés à deux concepts, il doit présenter deux requêtes distinctes au système. Il peut simplement spécifier des conditions pour les attributs de concepts qui sont dans le chemin de la requête. Le chemin à travers lequel un utilisateur parcourt la hiérarchie de l'ontologie de domaine pour atteindre les termes de la requête est appelé le chemin de la requête. Par exemple, dans la requête ci-dessus, les conditions qui ont été définies pour les attributs Nom, Email, *Field, Sexe* et *âge* sont touts liées à des concepts dans le chemin de la requête.

Dans le processus de construction de la requête, l'utilisateur traverse la hiérarchie du modèle de représentation uniforme interne de l'ontologie de domaine afin de choisir les termes de sa requête. L'utilisateur interagit avec le système à travers l'interface utilisateur graphique « GUI » (Graphical User Interface) dans la veu « *Query Model* » du système (voir Figure 4.7). L'interface graphique doit afficher l'ontologie de domaine de sorte que l'utilisateur puisse trouver les termes de sa requête facilement et rapidement.

La manière dont on a développé l'interface graphique et l'affichage de la hiérarchie du modèle de représentation d'ontologie de domaine pour l'utilisateur, joue un rôle important dans la création précise et exacte de la requête utilisateur. La première interface graphique affiche une liste des ontologies de domaine. L'utilisateur choisit une ontologie de domaine lié à sa requête. Ensuite, l'interface affiche une hiérarchie de concepts de l'ontologie choisie. Pour une requête, l'utilisateur selectionne un des concepts, qui sera la racine de sa requête. Nous l'appelons **C**. Puis dans l'étape suivante les informations liées à **C** sont affichées :

- La définition détaillée du concept **C**.
- Les sous-concepts (enfants) de **C** dans la hiérarchie.
- Les attributs de **C**.
- Les relations et Co-domaine dont **C** est le domaine.

La Figure 4.7 fournit une illustration partielle de cette information sur le concept « Professor ».

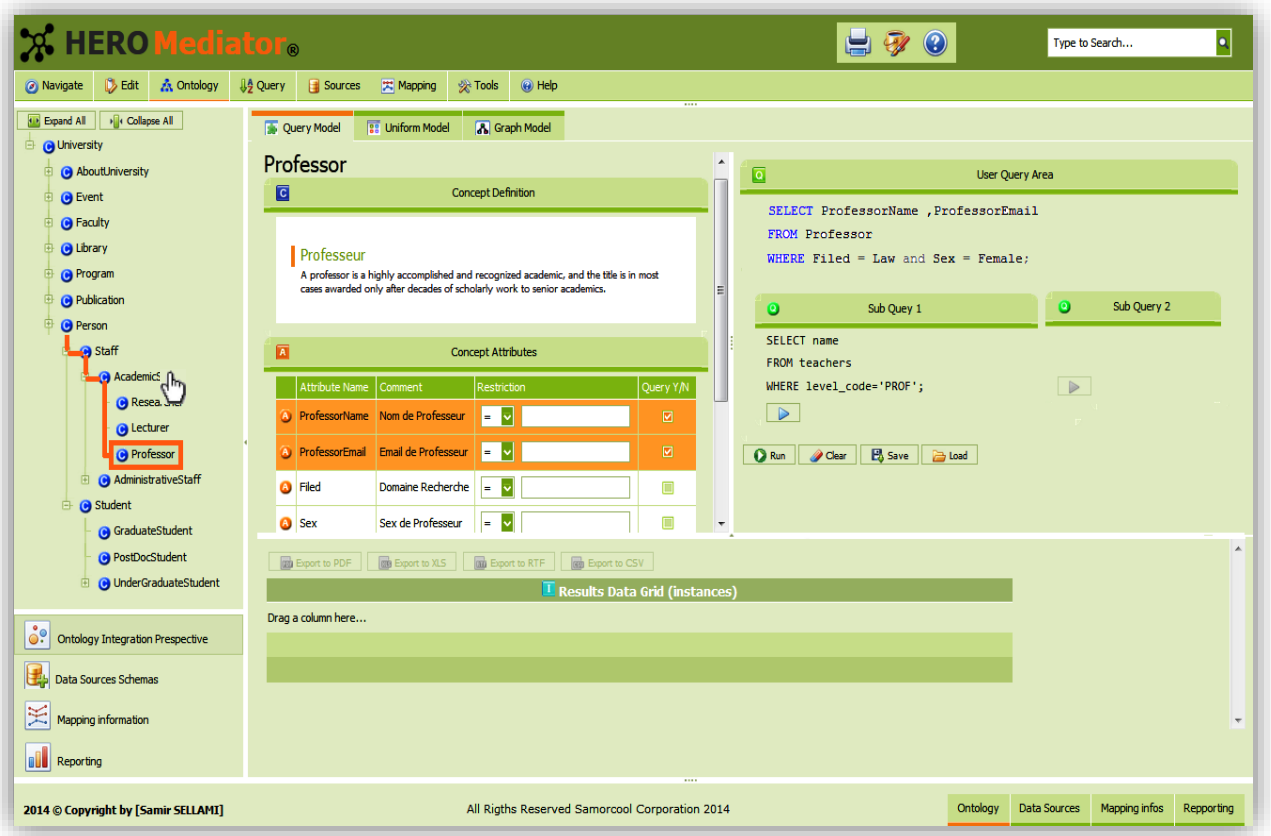


Fig. 4.7 – Exemple de création de requête sur le concept «Professor» de l'ontologie de domaine.

Dans cette étape :

- Si la requête a une condition ou une contrainte sur les attributs de **C**, alors l'utilisateur tappe la valeur de la contrainte des attributs dans les champs de "Restrictions" après la sélection de l'opérateur SQL *(=,<>,>,<,>=,<=, Between, Like or IN)* et ;
- Si ce concept est un concept de requête dans laquelle un utilisateur a besoin d'interroger ses attributs alors il coche la case à cocher "Query Y/N".

De cette manière, le module de création de requête obtient le chemin de la requête et la crée. Ce dernier ne consiste pas en des valeurs des attributs, mais seulement les noms des attributs. Il y'a juste un concept de requête dans le chemin de requête, ce qui signifie que l'utilisateur peut simplement interroger les attributs d'un concept (comme le montre la Figure 4.7 ci-dessus). Par exemple, le chemin d'accès pour la requête mentionnée plus haut est : University → Person → Staff → AcademicStaff → Professor (*Field, Sex, Age, Name?, Email?*)

En plus de son assistance dans la création de la requête utilisateur, HERO-Med offre des opérations de manipulation des requêtes telles que :

- *Run :* pour exécuter la requête afin d'avoir une réponse sur l'ensemble des sources de données connectées au médiateur.
- *Clear* : pour réinitialiser l'interface de création des requêtes et effacer la requête en cours, afin de définir une nouvelle requête.
- *Save :* pour sauvegarder la requête en cours sous format texte (ex : " myQry.txt "). Les requêtes sauvegardées peuvent être chargées ultérieurement. (voir Figure 4.8).
- *Load :* pour ouvrir une requête déjà sauvegardée, ou un fichier de requête de format texte (ex : " myQry.txt ").

Fig. 4.8 – GUI et exemple de sauvegarde de la requête utilisateur.

4.1 Implémentation des Translators

Après la création de la requête utilisateur à partir des termes de l'ontologie de domaine, le translator la traduit au langage de requêtes des sources de données locales. Pour chaque terme des sources de données, il existe des informations de mapping dans le « Schéma de Mapping » qui relie les éléments de l'ontologie domaine avec les éléments des sources de données. Cette information est utilisée par le translator pour traduire la requête en sous-requêtes aux termes des sources de données sous-jacentes à travers le bouton « *Create and Reformulate User Query* » (Voir Figure 4.9). L'exemple de la Figure 4.9 illustre la translation de la requête sur le concept « *AcademicStaff* » en deux sous-requêtes (SubQry1, SubQry2) qui correspondent aux sources de données locales (Source de données 1, Source de données 2) :

- Requête globale: *SELECT * FROM AcademicStaff; (sur le domaine d'ontologie).*
- SubQry1: *SELECT * FROM teachers; (sur la Source de données1).*
- SubQry2: *SELECT * FROM instructors; (sur la Source de données 2).*

Dans notre Prototype, les requêtes des utilisateurs réécrites en termes des sources de données locales sont envoyées aux adaptateurs. L'utilisateur a la possibilité d'exécuter soit la requête globale pour obtenir les résultats sur l'ensemble des sources de données connectées au médiateur, ou exécuter une sous-requête séparément, pour obtenir des résultats relatifs à la source de données correspondante.

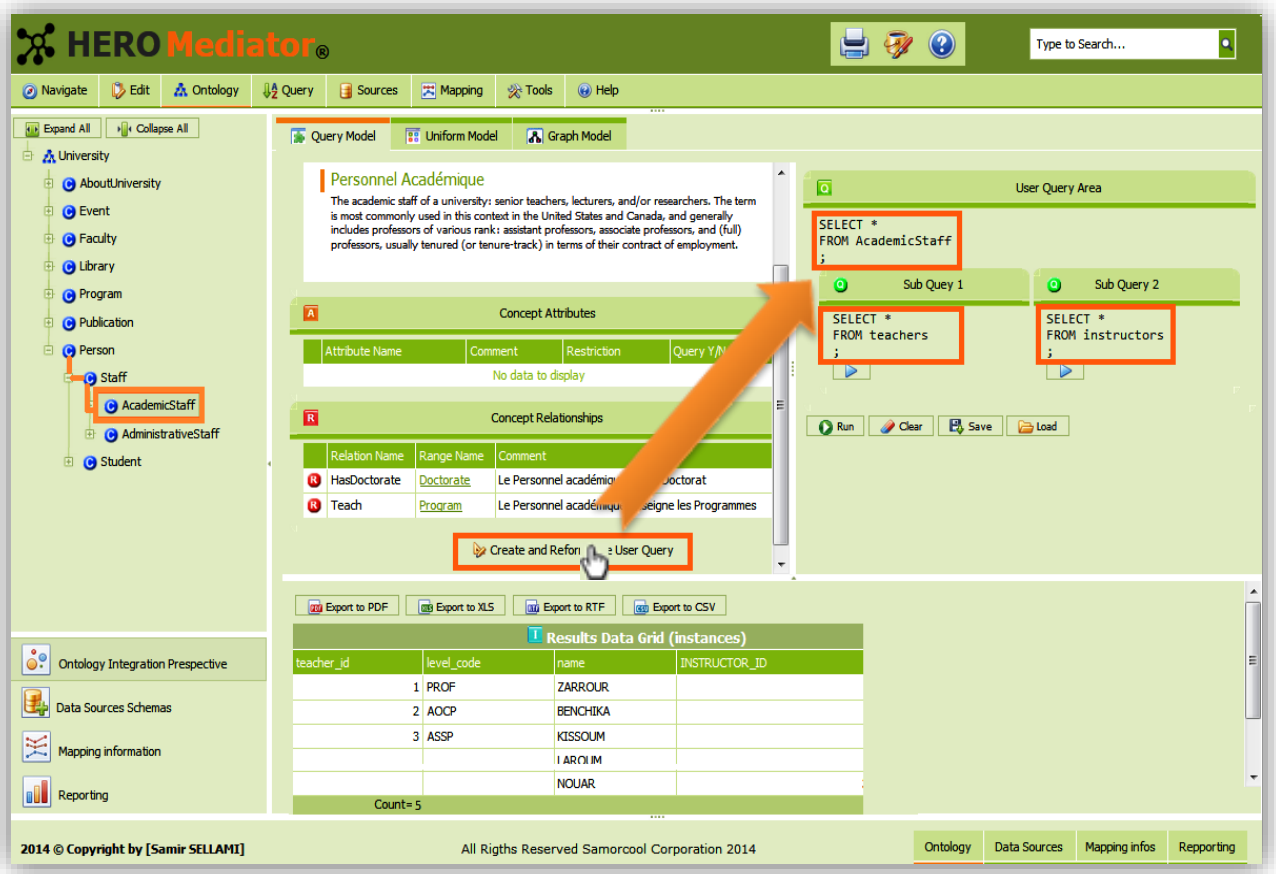


Fig. 4.9 – Interface de transtation de requête globale en deux sous-requêtes.

4.2 Implémentation des Adaptateurs

Les adaptateurs sont utilisés pour extraire les données à partir des sources de données locales présélectionnées. Un adaptateur est un module qui comprend l'organisation des données spécifiques [26] [56]. Il sait comment récupérer des données à partir de source sous-jacentes et cacher la structure des données spécifiques au reste du système d'intégration. Parmi les outils d'Adaptateurs disponibles en freeware nous citons :

1. W4F (http://cheops.cis.upenn.edu/W4F/)
2. XWRAP (http://www.cc.gatech.edu/projects/disl/XWRAPElite/)
3. DEByE (http://www.lbd.dcc.ufmg.br/~debye/)

Dans notre Prototype HERO-Med, nous exploitons les technologies offertes par la Plateforme .NET en matière d'accès aux données (ADO.Net et Entity Framwork).

4.3 Implémentation de l'Exportateur

Enfin les données sont prêtes à être présentées à l'utilisateur. Dans cette étape, ces dernières sont visualisées à l'environnement de représentation des utilisateurs. Nous présentons les données dans des GRIDS de données et des Pages HTML. L'utilisateur par la suite peut exporter les résultats de ces requêtes sous différents formats (Voir Figure 4.10):

- *Export PDF :* permet d'exporter les résultats dans un format PDF (Portable Document Format).
- *Export XLS :* permet d'exporter les résultats dans un format EXCEL (Feuille Microsoft EXCEL).
- *Export RTF:* permet d'exporter les résultats dans un format RTF (Rich Text Format).
- *Export CSV:* permet d'exporter les résultats dans un format CSV (Comma-Separated Values).

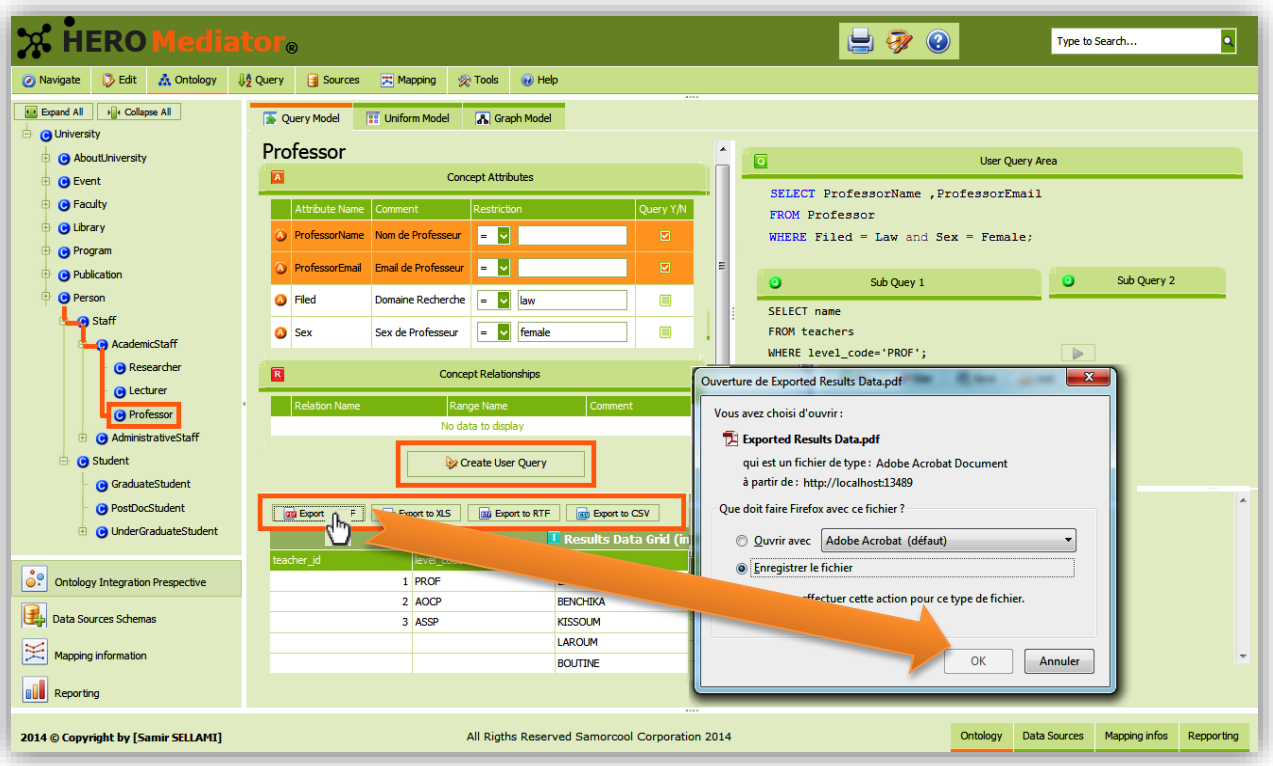


Fig. 4.10 – Reformulation de requête et exportation des resultas final.

HERO-Med permet aussi la mise à jour du modèle de représentation uniforme de l'ontologie de domaine à travers la vue « *Uniform Model* » (voir Figure 4.11), et la visualisation du modèle de graphe de l'ontologie de domaine à travers la vue « *Graph Model* » (voir Figure 4.12).

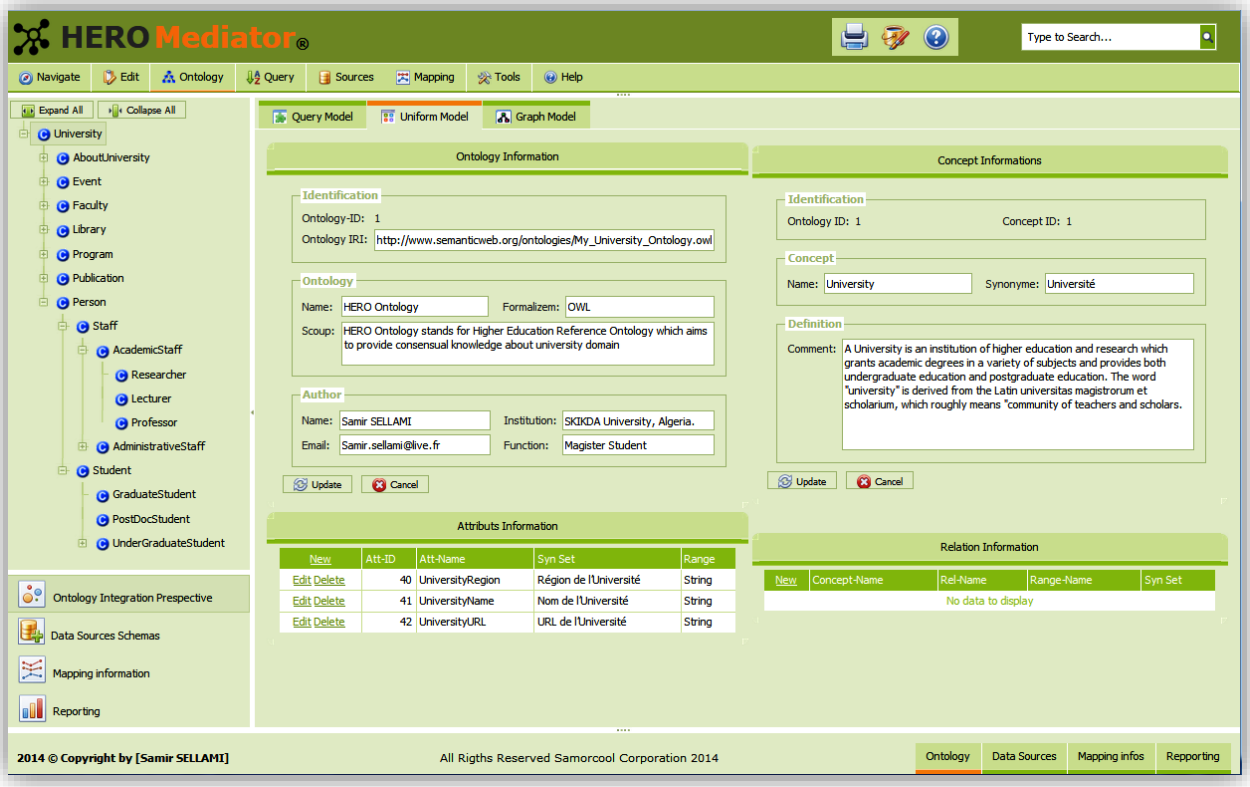


Fig. 4.11 – Interface de mis à jour du modèle de l'ontologie de domaine.

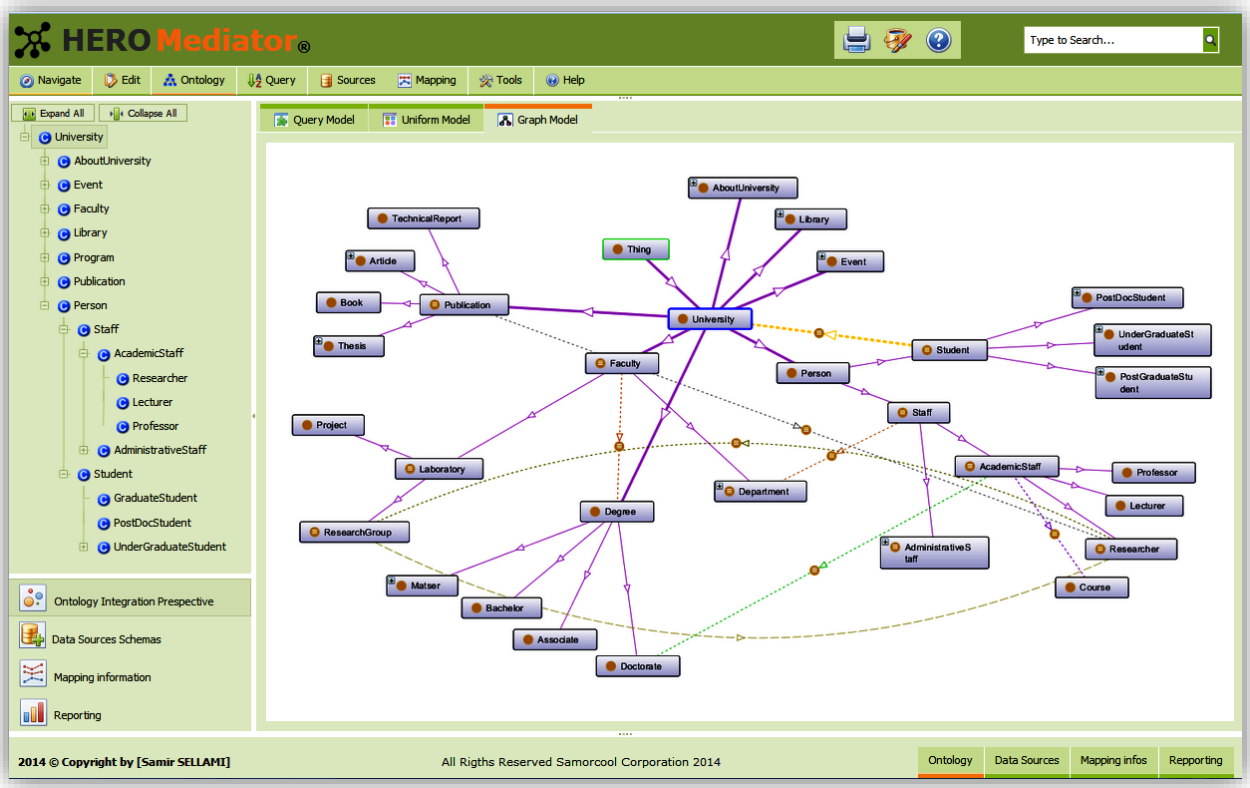


Fig. 4.12 – Interface de visualisation du modèle de graphe de l'ontologie de domaine.

5. Module des Sources de données

Pour évaluer notre approche d'intégration à base ontologique, nous avons choisi deux sources de données relationnelles hétérogènes conçues indépendamment par différents concepteurs (universités) à des fins applicatives distinctes. Il peut y avoir différents points de vue sur le même concept, à cause des distinctes interprétations des objets du monde réel. Ces deux sources sont implémentées dans des SGBD différents :

- **Source de données 1:** implémentée dans SGBD Microsoft SQL Server (Express Edition 2012) [22].
- **Source de données 2 :** implémentée dans SGBD Oracle Data base (Express Edition 11g) [23].

Les Scripts DDL (*Data Definition Language*) de création de ces deux sources de données sont présentés en détails dans l'Annexe B. Du fait que les deux sources ont le même type de structure de données relationnelles (voir Figure 4.13), l'hétérogénéité structurelle n'est pas traitée. L'hétérogénéité sémantique, par contre, présente un défi majeur dans le processus d'intégration.

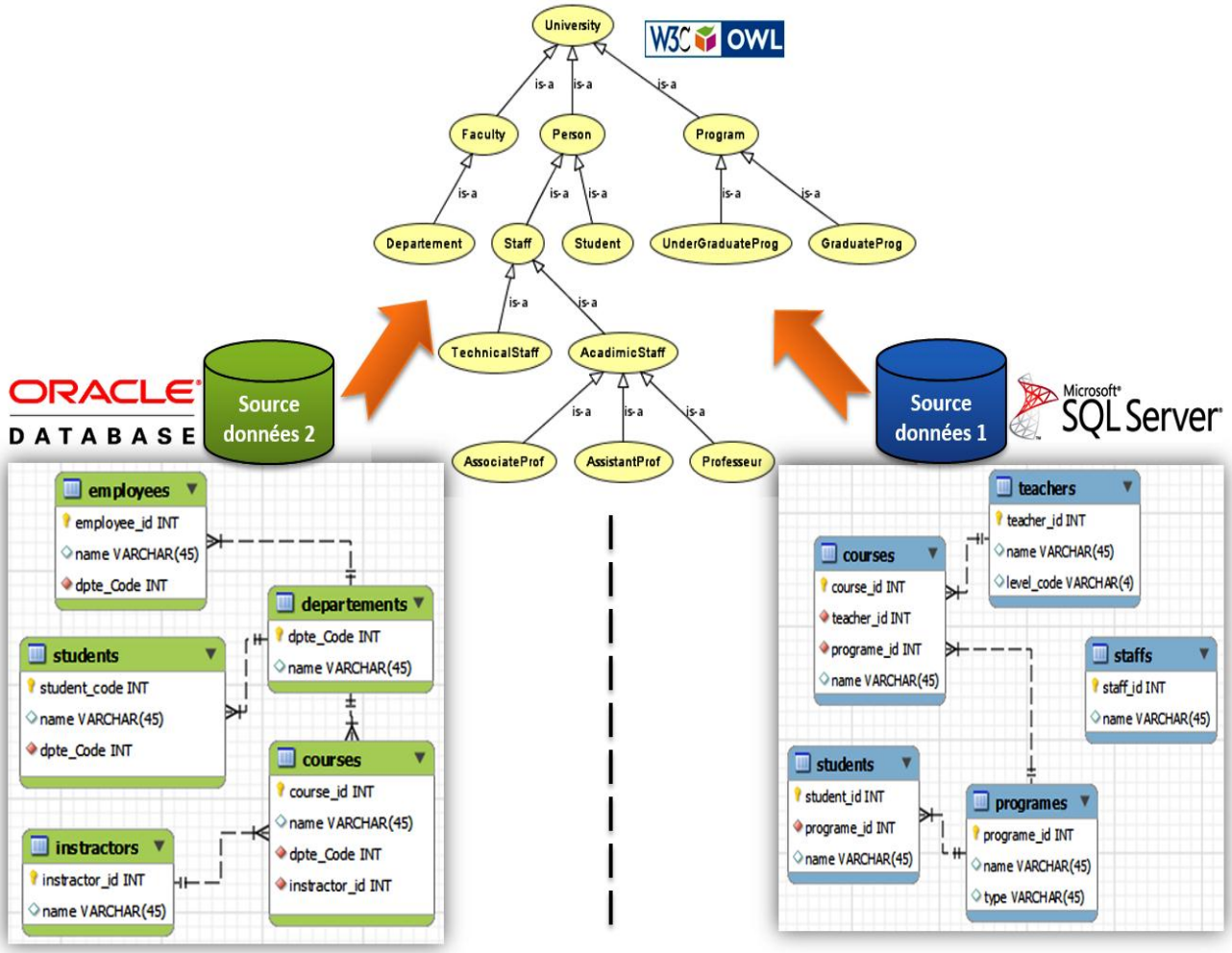


Fig. 4.13 – Exemple de sources de données hétérogènes.

[22] http:// www.microsoft.com/sql
[23] http://www.oracle.com

Pour pouvoir interroger des sources de données hétérogènes dans un environnement intégré, celles-ci doivent être connectées au médiateur. Le Prototype HERO-Med supporte la plupart des sources de données structurées relationnelles. Après avoir connecté les sources de données au médiator HERO-Med, ce dernier procède à l'extraction de leurs métadonnées et les stocke dans un schéma de mapping au niveau des tables métadonnées des sources (voir section 5, chapitre 3). La Figure 4.14 présente une vue des métadonnées extraites depuis les deux sources de données présentées ci-dessus :

- Les métadonnées en bleu : concernent la **Source de données 1**
- Les métadonnées en vert : concernent la **Source de données 2**

HERO-Med permet d'ajouter une nouvelle source de données au catalogue du système à travers le bouton « *Add Source* », ou retirer une source existante dans le catalogue en utilisant le bouton « *Remove Source* ».

Source-ID	Name
1	Datasource1
2	Datasource2

Table-ID	Source-ID	Table-Name
1	1	program
2	1	course
3	1	student
4	1	teacher
5	1	staff
6	2	department
7	2	instructor
8	2	student
9	2	course
10	2	employee

Column-ID	Table-ID	Column-Name
1	1	program_id
2	1	name
3	1	type
6	2	course_id
7	2	teacher_id
8	2	programe_id
9	2	name
10	3	student_id
11	3	name
12	3	programe_id
13	4	teacher_id
14	4	name
	4	level_code
16	5	staff_id
17	5	name

Column-ID	Table-ID	Column-Name
18	6	dpte_code
19	6	name
20	7	instructor_id
21	7	name
22	8	student_code
23	8	name
24	8	dpte_code
25	9	Course_id
26	9	Name
27	9	Dpte_code
28	9	instructor_id
29	10	employee_id
30	10	name
31	10	dpte_id

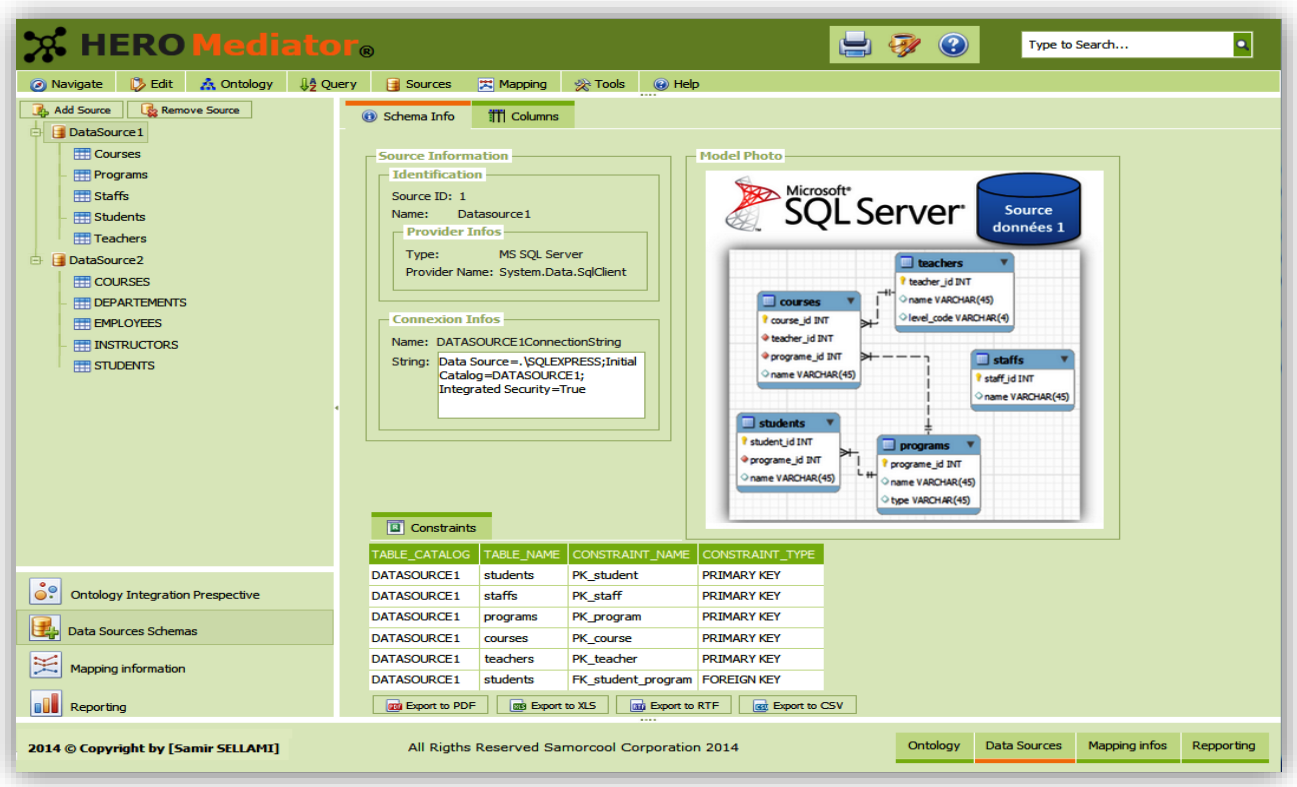


Fig. 4.14 – Interface des métadonnées des sources.

6. Module information de Mapping

6.1 Implémentation de module Mapping

Notre module de mapping est un outil graphique qui permet la mise en correspondance entre les termes des sources de données locales et ceux de l'ontologie de domaine. Ce module est implémenté dans le Framework .NET et Java Script en utilisant l'environnement Visual Studio 2012 (voir Figure 4.15). Les mappings peuvent être définis en fonction de l'interface graphique, en utilisant un schéma-vue de l'ontologie de domaine et les métadonnées des sources de données respectives. Pour cela, on doit juste comprendre la sémantique de l'interface graphique (par exemple une flèche reliant deux classes, ou deux Attributs), sans se soucier de la représentation logique des correspondances.

Le Prototype HERO-Med permet de créer trois types de mapping (voir Figure 4.15) :

- *Classe-à-Classe Mapping :* permet de créer un mapping entre les tables des sources des données et les concepts de l'ontologie de domaine.
- *Vue-à-Classe Mapping* : permet de créer un mapping entre une vue personnalisée et un concept de l'ontologie de domaine.
- *Attribut-à-Attribut Mapping:* permet de créer un mapping entre les colonnes des tables des sources de données et les attributs des concepts de l'ontologie de domaine.

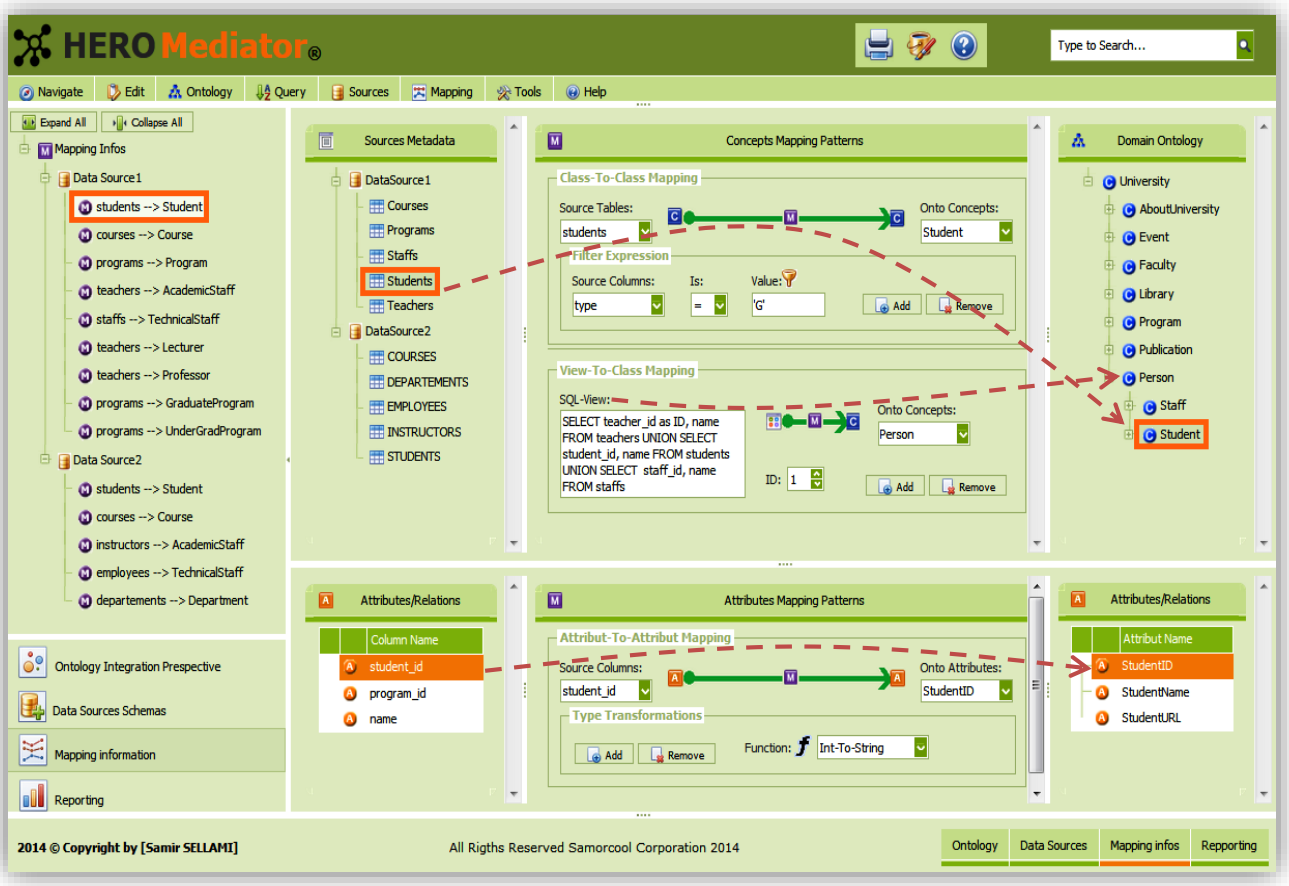


Fig. 4.15 – Interface de mapping du Prototype HERO-Med.

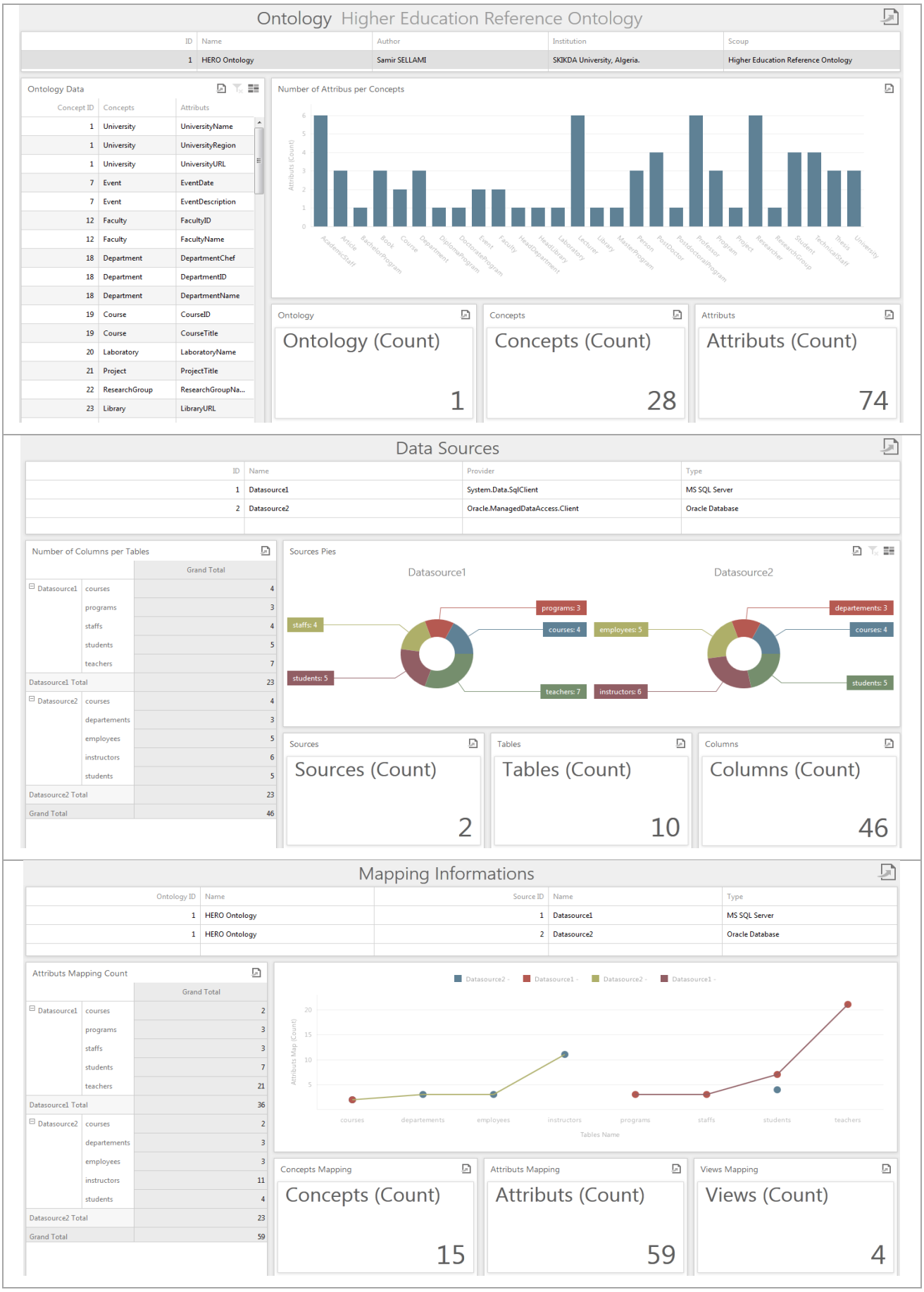


Fig. 4.16 – Tableaux de bord (dashboards) du Prototype HERO-Med.

6.2 Implémentation des Filtres

Les filtres peuvent être spécifiés pour les mapping de type « *Classe-à-Classe* » avec lesquels les instances de mapping sont limitées en fonction des valeurs de propriétés des colonnes de tables des sources de données. Les filtres facilitent le mapping dans le cas où il existe beaucoup de hiérarchisation au niveau des concepts de l'ontologie de domaine.

6.3 Implémentation des Convertisseurs

Pour le type de mapping « *Attribut-à-Attribut* » il est possible de définir des conversions via le convertisseur. Après l'extraction des réponses par les adaptateurs à partir de sources de données locales, elles sont envoyées au convertisseur. Ce dernier résout l'hétérogénéité de valeur de données. Les conflits de valeurs de données sont :

- Les Types (ex. entier ou caractère).
- Les Unités (ex. "kg" ou "gram").
- Format de valeur (ex. format date, format temps).
- Valeur de signe (ex. "$" ou "dollar").

Pour résoudre ce genre de conflits, le convertisseur utilise des fonctions de conversion. Dans notre Prototype, nous définissons des fonctions de conversion pour résoudre les conflits entre les valeurs de données telles que les entiers, les noms, les adresses URI, les Emails et les dates. Nous définissons un format standard pour ces valeurs.

7. Module Reporting

Ce module permet la visualisation des tableaux de bord (dashboards) pour les modules précédents : (Ontologie, Sources des données et informations du Mapping) voir Figure 4.16.

8. Conclusion

Dans ce chapitre, nous nous sommes concentrés sur la mise en œuvre de notre approche d'intégration des données hétérogènes à base ontologique. À cette fin, nous avons présenté une méthodologie pour construire les ontologies de domaine et nous avons créé une ontologie de domaine de l'université. Nous avons développé une interface graphique. Celle-ci est un ensemble de pages Web permettant de créer les requêtes utilisateurs, manipuler des sources de données hétérogènes, définir des mapping vers l'ontologie de domaine, visualiser les résultats de l'intégration finale et l'exporter au format souhaité.

Nous avons considéré deux sources de données locales relationnelles hétérogènes afin d'évaluer notre approche de mapping. Nous avons implémenté notre prototype avec le Framework .NET sur une Plateforme Web. Celle-ci assure l'ouverture et l'accessibilité qui sont deux caractéristiques nécessaires pour un système d'intégration de données. La qualité des résultats de mapping est en relation directe avec l'intégralité de l'ontologie de domaine.

Conclusion générale et Perspectives

« Plus s'étend et s'approfondie le champ de ma connaissance, plus s'aiguise la conscience de l'étendu de mon ignorance. »

Michel Beaud

Conclusion générale et perspectives

Sommaire

1 Conclusion 121
2 Contributions 121
3 Perspectives 123

1. Conclusion

L'intégration de données à pour objectif de combiner des sources de données autonomes, distribuées et hétérogènes afin d'obtenir une vue homogène et uniforme des données intégrées. Cependant, elle est posée à la fois à des problèmes structurels et des problèmes sémantiques. Les travaux concernant l'hétérogénéité structurelle sont relativement anciens et ont abouti à diverses approches permettant de la traiter dans le contexte des fédérations de bases de données ou des Multi-bases de données. L'hétérogénéité sémantique, par contre, reste la plus importante difficulté. Plusieurs systèmes récents d'intégration de sources de données hétérogènes utilisent les ontologies afin de résoudre les conflits sémantiques. Les principales limitations de ces systèmes sont soit l'absence d'automatisation en absence d'une ontologie de domaine partagée, soit l'absence d'autonomie (ou la faible autonomie) des sources dans le cas où une ontologie de domaine partagée est utilisée, et, dans tous les cas, Le maintien et la mise à jour de schéma global est très longue et coûteuse car un nombre important des sources de données sont impliquées et qui changent fréquemment. Dans ce travail, nous avons essayé de résoudre ces problèmes dans le contexte d'intégrations des sources de données relationnelles en utilisant un modèle de représentation d'ontologie de domaine spécifique et surmonté les problèmes d'hétérogénistes sémantiques entre les sources de données à travers un mapping LaV (local-as-view) entre ces sources et l'ontologie de domaine.

Dans cette dernière section nous présentons tout d'abord une synthèse de nos contributions et nous terminons par les différentes perspectives que nous avons dégagées pour nos travaux.

2. Contributions

Dans ce travail, nous avons proposé une approche d'intégration des données relationnelles a base ontologique de type virtuel (médiateur de données) permettant (1) une intégration automatique de sources de données, (2) une grande autonomie des sources. Les différentes contributions de ce travail sont les suivantes :

Présentation d'une nouvelle classification pour les systèmes d'intégration

Pour mieux comprendre les différentes approches et systèmes d'intégration, nous avons présenté un état de l'art. Cette étude nous a amené à présenter une nouvelle classification des approches d'intégration en se basant sur cinq critères orthogonaux suivants : (1) la représentation de données intégrées (approche médiateur ou entrepôt), (2) le sens de la mise en correspondance entre schéma global et schéma local (GaV, LaV ou Mixed), (3) la nature du processus d'intégration (manuelle, semi-automatique et automatique). (4) langages de représentation de données intégrées, et (5) niveaux d'abstraction. Cette classification nous a permis de présenter de façon synthétique les méthodes d'intégration existantes et de positionner précisément notre travail de l'intégration à base ontologique.

Nous avons de plus discuté les approches GaV et LaV. La plupart des études considèrent que l'approche LaV est flexible à l'ajout de nouvelles sources et assure le passage à l'échelle par apport à l'approche GaV, tandis que la réécriture de requêtes et plus facile dans l'approche GaV par apport à l'approche LaV.

Proposition d'une approche d'intégration des sources de données relationnelles

Nous avons proposé, une approche et une architecture système pour l'intégration des sources de données relationnelles structurées. Cette approche utilise des ontologies pour résoudre les conflits sémantiques entre les sources de données locales. Nous avons utilisé le modèle de représentation de l'ontologie de domaine pour la création de requêtes des utilisateurs. Il existe une ontologie de domaine spécifique pour chaque domaine d'application qui englobe la définition sémantique des concepts qui sont nécessaires pour fournir les requêtes des utilisateurs dans ce domaine d'application particulier. L'ontologie de domaine est modélisée dans un modèle de représentation uniforme interne au système. L'utilisateur peut naviguer dans l'ontologie de domaine et choisir les termes de sa requête, ensuite le système crée la requête de l'utilisateur. Nous supposons que chaque source de données contient des métadonnées qu'elle décrit. Suite à la création de la requête de l'utilisateur, le système choisi une source de données locale et les termes de la requête de l'utilisateur sont mis en correspondance avec ceux de la source de données. Ensuite, la requête de l'utilisateur est réécrite en utilisant les termes de la source de données locale. Les sous-requêtes provenant du processus de réécriture sont transformées aux langages de requête des sources de données, alors que les réponses à ces sous-requêtes sont extraites de sources de données correspondant par les adaptateurs. Enfin, les données sont, converties et présentées au format d'affichage commun à l'utilisateur.

Notre système met en œuvre une approche fondée sur les requêtes pour l'extraction et l'intégration de l'information, à partir de sources de données relationnelles, hétérogènes et distribuées. Le processus d'extraction et d'intégration dans le système proposé se compose de dix grandes tâches comme suit :

1. Transformation de l'ontologie de domaine à un modèle de représentation uniforme interne ;
2. Création de requête de l'utilisateur ;
3. Détermination des sources de données sous-jacentes avec le domaine de la requête ;
4. Mapping sémantique entre les termes de la requête et les termes des sources de données locales correspondantes ;
5. Réécriture de la requête utilisateur avec les termes correspondants des sources locales ;
6. Reformulation de la requête et la création d'un plan de requête ;
7. Traduction des sous-requêtes aux langages de requête des adaptateurs ;
8. Extraction des données ;
9. Conversion de valeurs de données ;
10. Exportation des données et leurs présentations au format de l'utilisateur ;

Notre approche d'intégration de données proposée couvre tous les niveaux d'abstraction des conflits d'hétérogénéité entre les sources de données locales. Le système applique :

- ***Une ontologie*** de domaine et un schéma de mapping relationnel entre les termes de chaque source et les concepts de l'ontologie de domaine, comme une solution pour résoudre les hétérogénéités sémantiques des schémas;
- ***Adaptateur*** comme solution pour résoudre les hétérogénéités syntaxiques des modèles ;
- ***Convertisseur*** comme solution pour résoudre les hétérogénéités de valeurs de données ;

Le système d'intégration de données proposé est extensible à n'importe quel domaine par l'ajout d'une ontologie de domaine spécifique au système. Cela signifie que pour pouvoir utiliser le système dans n'importe quel domaine d'application, nous devons développer et ajouter une ontologie de domaine (correspondant au domaine d'application) de système.

Validation de l'approche en implémentant le Prototype HERO-Med

Afin de valider notre proposition, nous l'avons prototypé. Cette implémentation est effectuée sur la Plateforme .NET en utilisant le langage ASP.Net, C# dans l'environnement Visual Studio 2012. Notre domaine d'application cible est celui de l'éducation supérieure et les universités. Notre Prototype HERO-Med (Higher Education Reference Ontology Mediator) nous permet d'intégrer automatiquement les sources de données hétérogènes de différentes universités, en offrant, sans aucune programmation, des possibilités d'accès ergonomiques et une interface graphique assiste l'utilisateur dans la création précise et exacte de sa requête. A cet effet, nous avons développé une ontologie de domaine spécifique dans le domaine des universités en utilisant le formalisme OWL, ensuite la représenter avec notre modèle de représentation uniforme interne. Nous avons évalué notre approche en utilisant deux sources de données relationnelles hétérogènes (MS SQL Server et Oracle Database). Les résultats de l'intégration étaient acceptables dans la mesure où l'ontologie de domaine englobe la terminologie des sources.

3. Perspectives

De nombreuses perspectives tant à caractère théorique que pratique peuvent être envisagées. Dans cette section nous présentons succinctement celles qui nous paraissent être les plus intéressantes.

L'Approche d'ontologies hybrides

Dans notre approche d'intégration nous avons utilisé une architecture d'ontologie de domaine unique, ce choix est motivé par le fait que les sources de données appartiennent au même univers et ont une vue similaire du domaine d'application (les universités dans notre cas). L'utilisation de l'architecture d'ontologies Hybride est très recommandée si on veut intégrer des sources appartenant à des différents domaines d'application.

Intégration des sources de données de différents types

Dans notre approche d'intégration les données sont de type structuré du fait qu'on s'intéresse à des bases de données relationnelles. Un prolongement naturel de notre travail d'intégration serait la prise en compte de données non structurées (images, multimédia, texte libre), de données semi-structurées (sources décrites par schéma XML, DTD), et les LOD (Linked Open Data). Les LOD ce sont des données sur le Web liées à d'autres sources et qui sont ouverts et accessibles à d'autres développeurs [115].

Requêtes complexes et syntaxe SPARQL

Dans notre approche les requêtes sont simples. Malgré que notre approche permet la création des requêtes correctes et précises sur un domaine, les utilisateurs ne peuvent pas poser des requêtes complexes, vu que la construction de la requête et sa structure sont basées sur une structure de requête interne au système et fondées sur des éléments de modèle de représentation uniforme interne de l'ontologie de domaine. Etendre notre approche afin de supporter des requêtes complexes (jointure entre concepts) est un point que nous envisageons de réaliser.

La syntaxe SQL utilisée pour traduction des requêtes posées permettra une interrogation parfaite des bases de données relationnelle. Cependant, afin de permettre une interrogation des différents types de données, nous envisageons de focaliser sur le langage SPARQL [116], qui a reçu le statut de « Recommendation Candidate » et qui sera très probablement le langage d'interrogation standard du Web Sémantique.

Bibliographie

[1] A. Elmagarmid, M. Rusinkiewicz and A. Sheth, (1998): "Management of Heterogeneous and Autonomous Database Systems". *Morgan Kaufmann Publishers*, 3eme édition. October 1998.

[2] T. Trâm, D. Ngoc, (2003): "Fédération de données semi-structurées avec XML". *Thèse de Doctorat en Informatique*, de l'Université de Versailles Saint-Quentin-en-Yvelines. Juin 2003.

[3] X. Baril, (2003): "Un modèle de vues pour l'intégration de sources de données XML: VIMIX". *Thèse de Doctorat en informatique*, de l'Université des sciences et techniques du Languedoc. Décembre 2003.

[4] L. Bellatreche, G. Pierra, D. Nguyen Xuan, et D. Hondjack, (2004) : "Intégration de sources de données autonomes par articulation a priori d'ontologies". A paraître dans : *Actes du XXIIème - Congrés INFORSID*, Biarritz, Pages 25-28. Mai 2004.

[5] A. RAHNI. A.Amidha, (2005): "Une approche médiateur d'intégration de sources de données hétérogènes et autonomes". Mémoire de stage effectué à *l'ENSMA*, Université de Poitiers. Juillet 2005.

[6] C. Goh, S. Bressan, E. Madnick, et M. D.Siegel, (1999): "Context interchange: New features and formalisms for the intelligent integration of information". *ACM Transactions on Information Systems*, 17(3), Pages 270–293, 1999.

[7] M. Jarke, M. Lenzerini, Y. Vassiliou, et P. Vassiliadis, (2000): "Fundamentals of Data Warehouses", Berling, *Springer Verlag*, 2000.

[8] C. Parent, S. Spaccapietra, (1996): "Intégration de bases de données: Panorama des problèmes et des approches". *Ingénierie des systèmes d'Information*, Volume 4, n° 3, Pages 333-358, 1996.

[9] S. Conrad, G. Saake, (1998): "Integrating heterogeneous databases". EDBT'98, *6th International Conference on Extending Database Technology*, Valencia, Spain, March 23-27, 1998.

[10] G. Salzano, H. Bestougeff, (1998): "Finding Related Information in Distributed Heterogeneous Systems". *In Proceedings of 16th International Conference* - Data Management in the Evolving Information Society (CODATA), New Delhi, November 1998.

[11] G. Falquet, M. Léonard, (1994) : "Intégration de concepts et intégration de bases de données". *Actes du Xème congrès INFORSID*, Aix en Provence 1994.

[12] E. Pitoura, (1995): "Extending an object-oriented programming language to support the integration of heterogeneous database systems". *In Proceedings of the 28th annual Hawaii International Conference on System Sciences*, Pages 707-716, 1995.

[13] R. Hull, (1997): "Managing semantic heterogeneity in databases: A theoretical perspective". *Proceedings of the Sixteenth ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems*, Pages 51–61, May 1997.

[14] S. Busse, R. Kutsche, U. Leser et H. Weber, (1999): "Federated Information Systems: Concepts, Terminology and Architectures". *Technical Report* n°99-9, Technical University of Berlin, 1999.

[15] W. Litwin, A. Abdellatif, A. Zeroual, B. Nicolas, Ph. Vigier, (1989): "MSQL: A Multidatabase Language". Pages 59-101, 1989.

[16] T. Landers, R.L. Rosenberg, (1982): "An overview of Multi-base". H.J. Schneider, editor, *Second International Symposium on Distributed Data Bases* (DDB 1982), Pages 153-84. 1982.

[17] C. Fankam, (2009): "OntoDB2 : un système flexible et efficient de Base de Données à Base Ontologique pour le Web sémantique et les données techniques". *Thèse de Doctorat en informatique*, ENSMA/Université de Poitiers, 2009.

[18] A.P. Sheth, J.A. Larson, (1990): "Federated database systems for managing distributed, heterogeneous, and autonomous databases". *ACM Comput. Survey*., 22(3), Pages 183-236. 1990.

[19] G.L G. Gomez, (2005): "Construction automatisée de l'ontologie de systèmes médiateurs. Application à des systèmes intégrant des services standards accessibles via le Web". *Thèse de Doctorat en informatique*, Université Paris XI Orsay, 2005.

[20] M.S. Hacid, C.Reynaud, (2004): "L'intégration de sources de données". *La Revue I3 : Information - Interaction - Intelligence,* Volume 4 n°2, 2004.

[21] P. Vassiliadis, A. Simitsis, P. Georgantas, M. Terrovitis,et S. Skiadopoulos, (2005): "A generic and customizable Framework for the design of etl scenarios". *Information Systems*, 30(7), Pages 492–525, 2005.

[22] S. Chen, B. Liu,et E.A. Rundensteiner, (2004): "Multiversion-based view maintenance over distributed data sources". *ACM Transactions on Database Systems*, 4(29), Pages 675–709, December 2004.

[23] M.Jarke, Y. Vassiliou, (1997): "Data warehouse quality design: A review of the DWQ Project". In Invited Paper, *2nd Conference on Information Quality*. Massachusetts Institute of Technology, Pages 299-313, Cambridge, 1997.

[24] J.Hammer, H. Garcia-Molina, J. Wi-dom, W. Labio, et Y. Zhuge, (1995): "The stanford data warehousing project". *IEEE Quarterly Bulletin on Data Engineering*; Special Issue on Materialized Views and Data Warehousing, 18(2), Pages 41–48, 1995.

[25] W.J. Labio, Y. Zhuge, J. L. Wiener,H. Gupta, H. Garca-Molina, et J. Widom, (1997): "The WHIPS Prototype for data warehouse creation and maintenance". *In Proceedings of SIGMOD*, Pages 557–559, 1997.

[26] S. Chawathe, H. Garcia-Molina, J. Hammer, K. Ireland, Y. Papakonstantinou, J. D. Ullman, et J. Widom, (1994) : "The TSIMMIS Project: Integration of heterogeneous information sources". *Proceedings of the 10th Meeting of the Information*. Processing Society of Japan, Pages 7–18. 1994

[27] M.R. Genesereth, A.M. Keller, et O.M. Duschka, (1997): "Infomaster: an information integration system". *In ACM SIGMOD International Conference on Management of Data*, Pages 539–542, 1997.

[28] A.Y. Levy, A. Rajaraman, et J. Ordille, (1996): "The World Wide Web as a collection of views: Query processing in the information manifold". *Proceedings of the International Workshop on Materialized Views*: Techniques and Applications (VIEW'1996), Pages 43–55, June 1996.

[29] E.Mena, V. Kashyap, A. P. Sheth, et A. Illarramendi, (1996): OBSERVER: "An approach for query processing in global information systems based on interoperation across pre-existing ontologies". *In Conference on Cooperative Information Systems*, Pages 14–25, 1996.

[30] Y. Arens, C. A. Knoblock, (1993): "Sims: Retrieving and integrating information from multiple sources". *Proceedings of the International Conference on Management of Data* (SIGMOD'1993), Pages 562–563, May 1993.

[31] C. Reynaud, G. Giraldo, (2003): "An application of the mediator approach to services over the Web". *Special track: Data Integration in Engineering*, Concurrent Engineering (CE'2003), Pages 209–216, July 2003.

[32] P.R. S. Visser, M. Beer, T. Bench Capon, B. M. Diaz, et M. J. R. Shave, (1999): "Resolving ontological heterogeneity in the kraft project". *10th International Conference on Database and Expert Systems Applications* (DEXA'99), Pages 668–677, September 1999.

[33] D. Nguyen Xuan, (2006): "Intégration de bases de données hétérogènes par articulation à priori d'ontologies : application aux catalogues de composants industriels". *Thèse de Doctorat en informatique*, ENSMA/Université de Poitiers, 2006.

[34] Y. Papakonstantinou, H. Garcia-Molina, et J. Widom, (1995): "Object exchange across heterogeneous information sources". *In 11th Conference on Data Engineering IEEE Computer*, Pages 251–260., 1995.

[35] Z. Ives, D. Florescu, M. Friedman, A. Levy, et D. Weld, (2004): "An adaptative query execution engine for data integration". *In Proceedings of the ACM SIGMOD International Conference on Management of Data*, Philadelphia, Pages 299-310, June 2004.

[36] M.C Rousset, P. Adjiman, P. Chatalic, F. Goasdoué, et L. Simon, (2006): "Somewhere in the semantic Web". Wiedermann, J., Tel, G., Pokorný, J., Bieliková,M.et *Stuller,J.,éditeurs: SOFSEM 2006 - Proceedings*, Pages 84-99. Springer 2006.

[37] W. Nejdl, B. Wolf, C. Qu, S. Decker, M. Sintek, A. Naeve, M. Nilsson, M. Palmér, et T. Risch, (2002): "Edutella: a p2p networking infrastructure based on rdf". Pages 604-615. 2002.

[38] L.L. Yan, R.J. Miller, L.M Haas, R. Fagin, (2001): "Data driven understanding and refinement of schema mappings". *In Proceedings of the 2001 ACM SIGMOD International conference on Management of data*. Santa Barabara, Californiana, United States, Pages 485-496. 2001.

[39] T. Milo, S. Abiteboul, B. Amann, O. Benjelloun, et F.D. Ngoc, (2003): "Exchanging intensional XML data". *Proc. of SIGMOD*, Pages 289-300. 2003.

[40] W. Nejdl, B. Wolf, S. Staab, et J. Tane, (2002): "Edutella: Searching and annotating resources within an rdf-based p2p network". *In Proceedings of the Semantic Web Workshop, 11th International World Wide Web Conference*, Honolulu, Hawaii, USA, May 2002.

[41] P. Bernstein, F. Giunchiglia, J. Mylopoulos, L. Serafini, et I. Zaihrayeu , (2002) : "Data Management for peer-to-peer computing: A vision". *Workshop on the Web and databases* (WebDB 2002), Pages 89-94, 2002.

[42] I. Tatarinov, Z. Ives, J. Madhavan, A. Havely, D. Suciu, N. Dalvi, X. Dung, Y. Dadiyska, G. Miklau, et P. Mork, (2003): "The Piazza Peer Data Management Project". *ACM SIGMOD*, Volume 32 Issue 3, Pages 47 - 52. September 2003.

[43] D. Beneventano, S. Bergamaschi, S. Castano, A. Corni, R. Guidetti, G. Malvezzi, M. Melchiori, et M. Vincini, (2000): "Information integration: The MOMIS project demonstration". *In The VLDB Journal*, Pages 611–614, 2000.

[44] P. McBrien, A. Poulovassilis, (2003): "Data Integration by Bi-Directional Schema Transformation Rules". *19th International Conference on Data Engineering*, Pages 227-238, Bangalore, India, March 5-8, 2003.

[45] M. Boyd, P. McBrien, et N. Tong, (2002): "The AutoMed schema integration repository". *In Proceedings of BNCOD02, Springer Verlag LNCS*, 2405, Pages 42-45. 2002.

[46] M. Friedman, A. Levy, et T. Millstein, (1999): "Navigational plans for data integration". *Proc. of the 16th National Conference on Articial Intelligence* (AAAI), Pages 67-73. 1999.

[47] M. Friedman, D.S. Weld, (1997): "Efficiently Executing Information Gathering Plans". *In 15th International Joint Conference On Artificial Intelligence*, Pages 785-791, Nagoya, Japan, 1997.

[48] O. Etzioni, D. Weld, (1994): "A Softbot-Based Interface to the Internet". *Communications of the ACM*. Volume 37 Issue 7, Pages 72-76, July 1994.

[49] C. Delobel, M.C Rousset, (2001): "A uniform approach for querying large tree-structured data through a mediated schema". *In International Workshop on Foundations of Models for Information Integration* (FMII), 2001.

[50] B. Amann, A. Michard, (2001): "C-Web functional specification (v1.0)", 2001. http://cweb.inria.fr/Resources/C-Webtm, 23 May 2001.

[51] P. Bernstein, (2003): "Applying model management to classical meta data problems". *In Proceedings of the Conf.* on Innovative Database Research (CIDR'03).

[52] G. Wiederhold, (1992): "Mediators in the architecture of future information systems". *In Journal Computer IEEE Computer Society*, Volume 25 Issue 3, Pages 38-49, Press Los Alamitos, CA, USA, March 1992.

[53] M. Lenzerini, (2002): "Data integration: a theoretical perspective". *In PODS '02: Proceedings of the twenty-first ACM SIGMOD-SIGACT-SIGART symposium on principles of database systems*, Pages 233-246, New York, USA, ACM Press. 2002.

[54] D. Schenck, P. Wilson, (1994): "Information modeling the express way". *Oxford University Press*, USA, January 6. 1994.

[55] Y. Breitbart, A. Silberschatz, et G.R. Thompson, (1990): "Reliable transaction Management in a multidatabase system". *In Proceedings of the ACM SIGMOD International Conference on Management of Data*, Pages 215–224, 1990.

[56] R. Fileto, (2001): "Issues on Interoperability and Integration of Heterogeneous Geographical Data". *In Proceedings of the Third Brazilian Symposium on GeoInformatics*. Pages 129-136. 2001.

[57] P. Merle, C. Gransart, et J.M. Geib, (1996): "Corba script and corba Web: A generic object-oriented dynamic environment upon corba". *In In Proceedings of TOOLS Europe 96*, Pages 97–112, Paris, France, 1996.

[58] S. Bergamaschi, S. Castano, S. De Capitani di Vimercati, S. Montanari, M. Vincini, (1998): "Exploiting Schema Knowledge for the Integration of Heterogeneous Sources". *In Convegno Nazionale Sistemi di Basi di Dati Evolute* (SEBD98), Pages 103–120, Ancona, Italy, 1998.

[59] M. Uschold, (1996): "Building ontologies: Towards a unified methodology". *In 16th Annual Conference of the British Computer Society Specialist Group on Expert Systems*, Cambridge, UK. December 1996.

[60] J. Charlet, B. Bachimont, et R. Troncy, (2005): "Ontologies pour le Web sémantique". *La revue I3: Information - Interaction - Intelligence*, Volume 5 n°1, 2005.

[61] A. Miller, (1995): "Wordnet - A lexical data base for english". *Communications of the ACM*, 38(11), Pages 39–41, November 1995.

[62] D. Beneventano, S. Bergamaschi, (2004): "The MOMIS methodology for integrating heterogeneous data sources". *IFIP 18th World Computer Congress Topical* Sessions 22–27, Pages 19-24, Toulouse, France, August 2004.

[63] G. Pierra, (2002): "Un modèle formel d'ontologie pour l'ingénierie, le commerce électronique et le Web sémantique: Le modèle de dictionnaire sémantique PLIB". *Journées Scientifique WebSemantique*, Paris, 2002.

[64] F. Goasdoué, V. Lattès, et M.C. Rousset, (2000): "The use of carin language and algorithms for information integration: The PICSEL system". *International Journal of Cooperative Information Systems (IJCIS)*, Volume 9 n° 4: Pages 383–401, December 2000.

[65] S. Decker, M. Erdmann, D. Fensel, et R. Studer, (1999): "Ontobroker: Ontology-based access to distributed and semi-structured information". *In DS8*, Pages 351–369, 1999.

[66] M.F. Lopez, (1999): "Overview of methodologies for building ontologies". *In Proceedings of the IJCAI-99 Workshop on Ontologies and Problem Solving Methods KRR5*, Stockholm Sweden, August 2 1999.

[67] N.F Noy, et D.L. McGuinness, (2001): "Ontology development 101: A guide to creating your first ontology". *Technical report*, SMI-2001-0880, 2001.

[68] S. Bechhofer, F.V Harmelen, J. Hendler, I. Horrocks, D.L McGuinness, P.F. Patel-Schneider, et L. Stein, (2004): "Owl Web Ontology Language Reference". *W3C*, http://www.w3.org/TR/owl-ref/, February 2004.

[69] G. Pierra, (2006): "Context-explication in conceptual ontologies: PLIB ontologies and their use for industrial data". *In Journal of Advanced Manufacturing Systems* (JAMS), 2006.

[70] H. Wache, T. Vögele, U. Visser, H. Stuckenschmidt, G. Schuster, H. Neumann, et S. Hübner, (2001): "Ontology-based integration of information - a survey of existing approaches". *Proceedings of the International Workshop on Ontologies and Information Sharing*, Pages 108–117, August 2001.

[71] R. Pottinger, A.Y. Levy, (2000): "Minicon: A Scalable Algorithm for answering queries using Views". In *Proceedings VLDB 2000 of 26th International Conference on Very Large Data Bases*, Cairo, Egypt,

Pages 484-495. Morgan Kaufmann. September 10-14 2000.

[72] J.D Ullman, (1997): "Information integration using logical views". *In Proceedings of 6th International Conference Delphi*, Pages 19-40, Greece, January 8–10, 1997.

[73] A. Levy, D. Srivastava, T. Kirk, (1995): "Data Model and Query Evaluation in Global Information Systems". *Journal of Intelligent information Systems*. Volume 5, Pages 121-143. 1995.

[74] V.S. Subrahmanian, S. Adali, A. Brink, R. Emery, J. J. Lu, A. Rajput, T. J. Rogers, R. Ross, C. Ward, (1995): "HERMES: A heterogeneous reasoning and mediator system". *Technical Report*. Univ. of Maryland 1995.

[75] A.R Hurson, M.W Bright, (1991): "Multidatabase systems: An advanced concept in handling distributed data". *Advances in Computers*, Volume 32, Pages 149-200. 1991.

[76] W.F. Cody, L.M. Haas, W. Niblack, M. Arya, M.J. Carey, R. Fagin, M. Flickner, D. Lee, D. Petkovic, P.M. Schwarz, J.T. II, M.T. Roth, J.H. Williams, et E.L. Wimmers, (1995): "Querying multimedia data from multiple repositories by content: the garlic project". *In VLDB Conference*, Pages 17-35. 1995.

[77] R.J. Bayardo, et al., (1997): InfoSleuth: "Agent-based semantic integration of information in open and dynamic environments". *In Proceeding SIGMOD* '97, Volume 26 Issue 2, Pages 195-206. June 1997.

[78] M. Arenas, V. Kantere, A. Kementsietsidis, I. Kiringa, R. Miller, et J. Mylopoulos, (2003): "The Hyperion Project : From data integration to data coordination". *ACM SIGMOD Record*, Volume 32 Issue 3, Pages 53 - 58. September 2003.

[79] B. Amann, C. Beeri, I. Fundulaki, et M. Scholl, (2002): "Ontology-based integration of xml Web resources". *In International Semantic Web Conference* (ISWC), Pages 117-131, Sardinia, Italy, 2002.

[80] B. Amann, C. Beeri, I. Fundulaki, et M. Scholl, (2002): "Querying xml sources using an ontology-based mediator". *In CoopIS*,Volume 2519. Pages 429-448. 2002.

[81] T.R. Gruber, (1993): "Format ontology in conceptual analysis and knowledge representation". Chapter: Towards principles for the design of ontologies used for knowledge sharing. *Kluwer Academie Publishers*., 1993.

[82] N. Guarino, R. Poli, (1995): "Formal ontology in conceptual analysis and knowledge representation". *Special issue of the International Journal of Human and Computer Studies*, Volume 43 Issue 5-6, Pages 625-640. Nov-Dec 1995.

[83] S. Jean, G. Pierra, et Y. Ait-Ameur, (2007): "Domain Ontologies: a Database-Oriented Analysis". *In International Conferences WEBIST 2005 and WEBIST* 2006 Revised Selected Papers, Pages 238-254, Springer Berlin Heidelberg, 2007.

[84] D. Brickley, et R.Guha, (2002): "Rdf vocabulary description language 1.0: Rdf-schema". *W3C*, http://www.w3.org/TR/rdf-schema/, 2002.

[85] D. Connolly, F.V Harmelen, I. Horrocks, D.L McGuinness, P.F. Patel-Schneider, et L.A. Stein, (2001): "Daml+Oil reference description". *W3C*, http://www.w3.org/TR/daml+oil-reference, March 2001.

[86] S.Sellami, F.Benchikha (2013): "Ontology-Based Data Integration: Approach and System Architecture". *In Proceedings of the 2nd Conference on Theoretical and Applicative Aspects of Computer Science* CTAACS'13, 25/26 Nov 2013, Skikda, Algeria, Pages 117–129.

[87] IS0-13584-42, (1998): "Industrial Automation Systems and Integration Parts LIBrary Part 42: Description methodology: Methodology for Structuring Parts families". *Technical report*, ISO, 1998.

[88] O. Corby, R. Dieng-Kuntz, C. Faron-zucker, (2004): "Querying the semantic Web with the CORESE search engine". *In Proc. of the 16th European Conference on Artificial Intelligence* (ECAI'2004), Sub conference PAIS'2004, Pages 705-709, Valencia, August 2004.

[89] S. Jean, (2007): "OntoQL : un langage d'exploitation des bases de données à base ontologique". *Thèse de Doctorat*, LISI/ENSMA et Université de Poitiers. 2007. http://www.lisi.ensma.fr/ftp/Pub/documents/thesis/2007-thesis-jean.pdf, 2007.

[90] S.Sellami (2014): "Ontology-Based Integration for Disaster Management: Approach and System Architecture". *In Proceedings of the 1st International Conference on Information and Communication Technologies for Disaster Management* ICT-DM'14, 24/25 March 2014, Algiers, Algeria, Pages 139–140.

[91] D. Fensel, F.V Harmelen, I. Horrocks, D.L McGuinness, et P.F. Patel-Schneider, (2001): "OIL: an ontology infrastructure for the semantic Web". *Intelligent Systems, IEEE*, Volume: 16, Issue: 2, Pages: 38-45. Mar-Apr 2001.

[92] C. Fankmn, Y. Ait-Ameur, et G.Pierra, (2007): "Exploitation of ontology languages for both reasoning and persistency purposes: mapping PLIB, OWL and flight ontology models". *In J. Filipe, J. Cordeiro, B. Encarnaçao, and V. Pedrosa, editors, Third International Conference on Web Information Systems and Technologies* (WEBIST'07), Pages 254-262. INSTICC Press, March 2007.

[93] H. Dehainsala, (2007) : "Explicitation de la sémantique dans les bases de données : Le modèle OntoDB de bases de données à base ontologique". *Thèse de Doctorat en informatique*, LISI/ENSMA et Université de Poitiers. 2007.

[94] S. Alexaki, V. Christophides, G. Karvounarakis, D. Plexousakis, et K. Tolle, (2001) : "The ICS-FORTH RDFSuite: Managing Voluminous RDF Description Bases". *In Proceedings of the 2nd International Workshop on the Semantic Web*, Pages 1–13. 2001.

[95] B. McBride, (2001): "Jena: Implementing the RDF Model and Syntax Specification". *In Proceedings of the 2nd International Workshop on the Semantic Web*. Hong Kong, China. 1 May 2001.

[96] G. Pierra, H. Dehainsala, Y. Aït-Ameur, et L. Bellatreche,(2005): "Base de Données à Base Ontologique :principes et mise en œuvre". *Ingénierie des systèmes d'Information*, Volume 10/2, Pages 91-115. 2005.

[97] H. Dehainsala, G. Pierra, et L. Bellatreche, (2007) : "OntoDB : An Ontology-Based Database for Data Intensive Applications". *In Proceedings of the 12th International Conference on Database Systems for Advanced Applications* (DASFAA'07), Volume 4443 of Lecture Notes in Computer Science, Pages 497–508. Springer. 2007.

[98] M.J. Park, J.H. Lee, C.H. Lee, J. Lin, O. Serres, et C.W. Chung, (2007): "An Efficient and Scalable Management of Ontology". *In Proceedings of the 12th International Conference on Database Systems for Advanced Applications* (DASFAA'07), Volume 4443 of Lecture Notes in Computer Science, Pages 975–980. Springer. 2007.

[99] Z. Pan, J. Heflin, (2003): "DLDB: Extending Relational Databases to Support Semantic Web Queries". *In Proceedings of the 1st International Workshop on Practical and Scalable Semantic Systems* (PSSS'03), Pages 109–113. 2003

[100] L. Ma, Z. Su, Y. Pan, L. Zhang, et T. Liu, (2004): "RStar: an RDF storage and query system for enterprise resource Management". *In Proceedings of the 13th ACM International Conference on Information and Knowledge Management* (CIKM'04), Pages 484 – 491. 2004.

[101] E. Bozsak, et al., (2002): "KAON: Towards a Large Scale Semantic Web". *In Proceedings of the 3rd International Conference on E-Commerce and Web Technologies* (EC-WEB'02), Pages 304–313, London, UK. Springer Verlag. 2002.

[102] S. Harris, et N. Gibbins, (2003): "3Store: Efficient bulk RDF Storage". *In Proceedings of the 1st International Workshop on Practical and Scalable Semantic Systems* (PPP'03), Pages 1–15. 2003.

[103] K Stoffel, M. Taylor , et J. Hendler, (1997): "Efficient Management of Very Large Ontologies". *In Proceedings of the 14th National Conference on Artificial Intelligence and 9th Innovative Applications of Artificial Intelligence Conference* AAAI'97/IAAI'97, Pages 442–447. 1997.

[104] C. P. de Laborda, et S. Conrad, (2006): "Database to Semantic Web Mapping Using RDF Query Languages". *In Proceedings of the 25th International Conference on Conceptual Modeling* (ER'06), Pages 241–254. 2006.

[105] C. Li, T.W. Ling, (2005): "An improved prefix-labeling scheme: A binary string approach for dynamic

ordered xml". *In Database Systems for Advanced Applications* (DASFAA), Volume 3453, Pages 125-137. Springer, 2005.

[106] B. Motik, I. Horrocks, et U.Sattler (2007): "Bridging the gap between owl and relational databases". *In WWW'07 Proceedings of the 16th international conference on World Wide Web*, Pages 807-816. ACM, 2007.

[107] R. Volz, S. Staab, et B. Motik, (2005): "Incrementally Maintaining Materializations of Ontologies Stored in Logic Databases". *Journal of Data Semantics II*, 3360, Pages 1-34, 2005.

[108] E. Rahm, P.A. Bernstein, (2001): "A survey of approaches to automatic schema matching". *In Proceedings of the International Conference on Very Large Databases*, Volume 10 Issue 4, Pages 334-350. December 2001.

[109] P. Shvaiko, J. Euzenat, (2005): "A survey of schema-based matching approaches". *In Journal on Data Semantics IV*, Pages 146–171. Lecture Notes in Computer Science, December 2005.

[110] B. Magnini, L. Serafini, M. Speranza, (2002): "Linguistic based matching of local ontologies". *In Proceedings of Working Notes of the AAAI-02 workshop on Meaning Negotiation*., Edmonton, 2002.

[111] S. Suwanmanee, D. Benslimane, P.A. Champin, et P. Thiran, (2005): "Wrapping and integrating heterogeneous databases with OWL". *In Proceedings of the Seventh International Conference on Enterprise Information Systems 1* (ICIES 2005), Pages 11-18, Florida, USA. 2005.

[112] A. Doan, J. Madhavan, P. Domingos, A. Halevy, (2004): "Learning to map between Ontologies on the Semantic Web". *In Proceedings of the international WWW Conference*, Pages 662-673, New York, USA. 2004.

[113] A.D. Preece, K. Ying Hui, W.A. Gray, P. Marti, T.J.M. Bench Capon, D.M. Jones, et Z. Cui, (1999): "The KRAFT architecture for knowledge fusion and transformation". *In Proceedings of ES99, the Nineteenth International Conference on Knowledge-Based Systems and Applied Artificial Intelligence* (SGES), Pages 23-38, Cambridge, December 1999.

[114] T. Vogele, H.S., et S.G. Busteran, (2003): "BUSTER: an information broker for the semantic Web". *Knstliche Intelligenz 3*, Pages 31–34, July 2003.

[115] C. Bizer, T. Heath, T. Berners-lee, (2009): "Linked data -The story so far". *In International Journal on Semantic Web and Information*, Volume 5(3): Pages 1-22,

[116] J. Perez, M. Arenas, et C. Gutierrez, (2006): "Semantics and complexity of sparql". *In the 5th International Semantic Web Conference* (ISWC) 2006, Pages 30-43, Athens, GA, November 5-9 2006.

Glossaire

AI	Artificial Intelligence
BaV	Both-as-View
BDBO	Bases de Données à Base Ontologique
BGLaV	Brigham Young University Global Local-as-View
BSU	Basic Semantic Unit
CIF	Constraint Interchange Format
DARPA	Defense Advanced Research Projects Agency
DATALOG	Data Logic
DB	Data Base
DBO	Données à Base Ontologiques
DDL	Data Definition Language
DISA	Data Interchange Standards Association
DL	Description Logics
DTD	Document Type Definition
DWQ	Data Warehouse Quality
GaV	Global-as-View
GDS	Global Distribution Systems
GLaV	Generalized Local-as-View
GVV	Global Virtual View
GUI	Graphic User Interface
HERMES	HEterogeneous Reasoning and MEdiator System
HERO-Med	Higher Education Reference Ontology Mediator
IDE	Integrated Development Environment
KRAFT	Knowledge Reuse and Fusion/Transformation
LaV	Local-as-View
LOD	Linked Open Data
MeSH	Medical Subject Heading
MOMIS	Mediator envirOnment for Multiple Information Sources
OBSERVER	Ontology Based System Enhanced with Relationships for Vocabulary hEterogeneity Resolution
OC	Ontologie Conceptuelle
OCC	Ontologie Conceptuelle Canonique
OEM	Objet Exchange Model
OG	Ontologie Globale

OIL	Ontology Inference Layer
OLAP	Online Analytical Processing
OL	Ontologie Linguistique
OO	Object Oriented
OWL	Ontology Web Language
PIAZZA	The Piazza Peer Data Management Project
PICSEL	Production d'Interface à base de Connaissance pour des Services En Ligne
PLIB	Parts Library (PLIB) - Norme ISO 13584
RDF	Resource Description Framework
RDFS	Resource Description Framework Schema
SHOE	Simple HTML Ontology Extensions
SIREN	Système d'Identification du Répertoire des ENtreprises
SIRET	Système d'Identification du Répertoire des Etablissements
SGBD	Système de Gestion de Base de Données
SIMS	Single Interface to Multiple Sources
SIG	Système d'Information Géographique
SIOC	Semantically-Interlinked Online Communities
SOA	Service-Oriented Architecture
STEP	STandard for Exchange Product model data)
TSIMMIS	The Stanford-IBM Manager of Multiple information Source
TALN	Traitement Automatique de la Langue Naturelle
URI	Uniform Resource Identifier
WHIPS	WareHouse Information Prototype at Stanford
XML	eXtensible Markup Language

Annexe A

Codage en OWL de l'Ontologie Université.

Cette annexe contient une partie de code OWL du L'ontologie de domaine l'Université en notation RDF/XML.

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

<!DOCTYPE rdf:RDF [
    <!ENTITY owl "http://www.w3.org/2002/07/owl#" >
    <!ENTITY xsd "http://www.w3.org/2001/XMLSchema#" >
    <!ENTITY rdfs "http://www.w3.org/2000/01/rdf-schema#" >
    <!ENTITY rdf "http://www.w3.org/1999/02/22-rdf-syntax-ns#" >
    <!ENTITY My_University_Ontology
"http://www.semanticweb.org/ontologies/2013/4/My_University_Ontology.owl#" >
]>

<rdf:RDF xmlns="http://www.semanticweb.org/ontologies/2013/4/My_University_Ontology.owl#"
     xml:base="http://www.semanticweb.org/ontologies/2013/4/My_University_Ontology.owl"
     xmlns:rdfs="http://www.w3.org/2000/01/rdf-schema#"

xmlns:My_University_Ontology="http://www.semanticweb.org/ontologies/2013/4/My_University_Ontology
.owl#"
     xmlns:owl="http://www.w3.org/2002/07/owl#"
     xmlns:xsd="http://www.w3.org/2001/XMLSchema#"
     xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#">
    <owl:Ontology
rdf:about="http://www.semanticweb.org/ontologies/2013/4/My_University_Ontology.owl">
        <rdfs:label xml:lang="en">HEROntotology</rdfs:label>
        <rdfs:comment xml:lang="en">HER ONTOLOGY AUTHOR INFORMATION:
Name: Samir SELLAMI
EMAIL: samir.sellami@live.fr
FUNCTION: magister student.
INSTITUTION: SKIKDA University, Algeria. </rdfs:comment>
        <rdfs:isDefinedBy xml:lang="en">HEROnto stands for Higher Education Reference Ontology
which aims to provide consensual knowledge about university domain</rdfs:isDefinedBy>
    </owl:Ontology>

    <!--
      ///////////////////////////////////////////////////////////////////////////////////////
     //                                    Object Properties                                  //
    ///////////////////////////////////////////////////////////////////////////////////////
     -->
    <owl:ObjectProperty rdf:about="&My_University_Ontology;Advisor"/>

    <owl:ObjectProperty rdf:about="&My_University_Ontology;BelongsTo"/>

    <owl:ObjectProperty rdf:about="&My_University_Ontology;ComposedOf"/>

    <owl:ObjectProperty rdf:about="&My_University_Ontology;EnrolledBy"/>

    <owl:ObjectProperty rdf:about="&My_University_Ontology;HasDgeree"/>

    <owl:ObjectProperty rdf:about="&My_University_Ontology;HasDoctorate"/>

    <owl:ObjectProperty rdf:about="&My_University_Ontology;HasMaster"/>

    <owl:ObjectProperty rdf:about="&My_University_Ontology;HeadOf"/>

    <owl:ObjectProperty rdf:about="&My_University_Ontology;ListOfCourse"/>

    <owl:ObjectProperty rdf:about="&My_University_Ontology;Offers"/>

    <owl:ObjectProperty rdf:about="&My_University_Ontology;PublishedBy"/>

    <owl:ObjectProperty rdf:about="&My_University_Ontology;StudyIn"/>

    <owl:ObjectProperty rdf:about="&My_University_Ontology;Teach"/>

    <owl:ObjectProperty rdf:about="&My_University_Ontology;WorkIn"/>
```

```
<!--
  ///////////////////////////////////////////////////////////////////////////////////
 //                                    Data properties                                //
///////////////////////////////////////////////////////////////////////////////////
 -->
<owl:DatatypeProperty rdf:about="&My_University_Ontology;ArticleAuthor">
    <rdfs:range rdf:resource="&xsd;string"/>
</owl:DatatypeProperty>

<owl:DatatypeProperty rdf:about="&My_University_Ontology;ArticlePublishDate">
    <rdfs:range rdf:resource="&xsd;dateTime"/>
</owl:DatatypeProperty>

<owl:DatatypeProperty rdf:about="&My_University_Ontology;ArticleTitle">
    <rdfs:range rdf:resource="&xsd;string"/>
</owl:DatatypeProperty>

<owl:DatatypeProperty rdf:about="&My_University_Ontology;Department-Name">
    <rdfs:range rdf:resource="&xsd;string"/>
</owl:DatatypeProperty>

<owl:DatatypeProperty rdf:about="&My_University_Ontology;EventDate">
    <rdfs:range rdf:resource="&xsd;dateTime"/>
</owl:DatatypeProperty>

<owl:DatatypeProperty rdf:about="&My_University_Ontology;EventDescription">
    <rdfs:range rdf:resource="&xsd;string"/>
</owl:DatatypeProperty>

<owl:DatatypeProperty rdf:about="&My_University_Ontology;FacultyName">
    <rdfs:range rdf:resource="&xsd;string"/>
</owl:DatatypeProperty>

<owl:DatatypeProperty rdf:about="&My_University_Ontology;LibraryURL">
    <rdfs:range rdf:resource="&xsd;string"/>
</owl:DatatypeProperty>

<owl:DatatypeProperty rdf:about="&My_University_Ontology;PersonEmail">
    <rdfs:range rdf:resource="&xsd;string"/>
</owl:DatatypeProperty>

<owl:DatatypeProperty rdf:about="&My_University_Ontology;PersonID">
    <rdfs:range rdf:resource="&xsd;string"/>
</owl:DatatypeProperty>

<owl:DatatypeProperty rdf:about="&My_University_Ontology;PersonName">
    <rdfs:range rdf:resource="&xsd;string"/>
</owl:DatatypeProperty>

<owl:DatatypeProperty rdf:about="&My_University_Ontology;ProfessorEmail">
    <rdfs:range rdf:resource="&xsd;string"/>
</owl:DatatypeProperty>

<owl:DatatypeProperty rdf:about="&My_University_Ontology;ProfessorName">
    <rdf:type rdf:resource="&owl;FunctionalProperty"/>
    <rdfs:range rdf:resource="&xsd;Name"/>
</owl:DatatypeProperty>

<owl:DatatypeProperty rdf:about="&My_University_Ontology;ProfessorResearchFiled">
    <rdfs:range rdf:resource="&xsd;string"/>
</owl:DatatypeProperty>

<owl:DatatypeProperty rdf:about="&My_University_Ontology;ProjectTitle">
    <rdfs:range rdf:resource="&xsd;string"/>
</owl:DatatypeProperty>

<owl:DatatypeProperty rdf:about="&My_University_Ontology;ResearcherEmail">
    <rdfs:range rdf:resource="&xsd;string"/>
</owl:DatatypeProperty>

<owl:DatatypeProperty rdf:about="&My_University_Ontology;ResearcherFiled">
    <rdfs:range rdf:resource="&xsd;string"/>
</owl:DatatypeProperty>

<owl:DatatypeProperty rdf:about="&My_University_Ontology;ResearcherName">
    <rdf:type rdf:resource="&owl;FunctionalProperty"/>
```

```
    <rdfs:range rdf:resource="&xsd;Name"/>
</owl:DatatypeProperty>

<owl:DatatypeProperty rdf:about="&My_University_Ontology;StudentID">
    <rdfs:range rdf:resource="&xsd;string"/>
</owl:DatatypeProperty>

<owl:DatatypeProperty rdf:about="&My_University_Ontology;StudentName">
    <rdfs:range rdf:resource="&xsd;Name"/>
</owl:DatatypeProperty>

<owl:DatatypeProperty rdf:about="&My_University_Ontology;StudentEmail">
    <rdfs:range rdf:resource="&xsd;string"/>
</owl:DatatypeProperty>

<owl:DatatypeProperty rdf:about="&My_University_Ontology;SubjectTitle">
    <rdfs:range rdf:resource="&xsd;string"/>
</owl:DatatypeProperty>

<owl:DatatypeProperty rdf:about="&My_University_Ontology;ThesisAuthor">
    <rdfs:range rdf:resource="&rdfs;Literal"/>
</owl:DatatypeProperty>

<owl:DatatypeProperty rdf:about="&My_University_Ontology;ThesisPublishDate">
    <rdfs:range rdf:resource="&xsd;dateTime"/>
</owl:DatatypeProperty>

<owl:DatatypeProperty rdf:about="&My_University_Ontology;ThesisTitle">
    <rdfs:range rdf:resource="&xsd;string"/>
</owl:DatatypeProperty>

<owl:DatatypeProperty rdf:about="&My_University_Ontology;UniversityCountry">
    <rdfs:range rdf:resource="&xsd;string"/>
</owl:DatatypeProperty>

<owl:DatatypeProperty rdf:about="&My_University_Ontology;UniversityName">
    <rdfs:range rdf:resource="&xsd;string"/>
</owl:DatatypeProperty>

<owl:DatatypeProperty rdf:about="&My_University_Ontology;UniversityURL">
    <rdfs:range rdf:resource="&xsd;string"/>
</owl:DatatypeProperty>

<rdf:Description rdf:about="&owl;topDataProperty">
    <rdfs:range rdf:resource="&xsd;string"/>
</rdf:Description>
<!--
  ///////////////////////////////////////////////////////////////////////////////////
 //                                     Classes                                     //
///////////////////////////////////////////////////////////////////////////////////
 -->
<Class rdf:about="&My_University_Ontology;AboutUniversity">
    <rdfs:subClassOf rdf:resource="&My_University_Ontology;University"/>
    <synonymes xml:lang="en">Overview, about-us, about-college</synonymes>
</Class>

<Class rdf:about="&My_University_Ontology;AcademicStaff">
    <equivalentClass>
        <Restriction>
            <onProperty rdf:resource="&My_University_Ontology;Teach"/>
            <someValuesFrom rdf:resource="&My_University_Ontology;Course"/>
        </Restriction>
    </equivalentClass>
    <rdfs:subClassOf rdf:resource="&My_University_Ontology;Staff"/>
    <synonymes xml:lang="en">Faculty, Academic-people, academician</synonymes>
</Class>

<Class rdf:about="&My_University_Ontology;AdministrativeStaff">
    <equivalentClass>
        <Restriction>
            <onProperty rdf:resource="&My_University_Ontology;Advisor"/>
            <someValuesFrom rdf:resource="&My_University_Ontology;Student"/>
        </Restriction>
    </equivalentClass>
    <rdfs:subClassOf rdf:resource="&My_University_Ontology;Staff"/>
    <synonymes xml:lang="en">Non-Academic-Staff, Management-staff</synonymes>
```

```
</Class>

<Class rdf:about="&My_University_Ontology;Article">
    <rdfs:subClassOf rdf:resource="&My_University_Ontology;Publication"/>
    <synonymes xml:lang="en">paper</synonymes>
    <hasKey rdf:parseType="Collection">
        <rdf:Description rdf:about="&My_University_Ontology;ArticleAuthor"/>
    </hasKey>
</Class>

<Class rdf:about="&My_University_Ontology;BachelorStudent">
    <rdfs:subClassOf rdf:resource="&My_University_Ontology;UnderGraduateStudent"/>
    <synonymes xml:lang="en">degree student</synonymes>
</Class>

<Class rdf:about="&My_University_Ontology;Book">
    <rdfs:subClassOf rdf:resource="&My_University_Ontology;Publication"/>
    <synonymes xml:lang="en">Work, Quid, Paper, Chapter</synonymes>
</Class>

<Class rdf:about="&My_University_Ontology;Course">
    <rdfs:subClassOf rdf:resource="&My_University_Ontology;Department"/>
    <synonymes xml:lang="en">subject, module</synonymes>
</Class>

<Class rdf:about="&My_University_Ontology;Dean">
    <equivalentClass>
        <Restriction>
            <onProperty rdf:resource="&My_University_Ontology;HeadOf"/>
            <onClass rdf:resource="&My_University_Ontology;Faculty"/>
            <maxQualifiedCardinality rdf:datatype="&xsd;nonNegativeInteger">1
            </maxQualifiedCardinality>
        </Restriction>
    </equivalentClass>
    <rdfs:subClassOf rdf:resource="&My_University_Ontology;AdministrativeStaff"/>
    <synonymes xml:lang="fr">Doyen</synonymes>
</Class>

<Class rdf:about="&My_University_Ontology;Department">
    <equivalentClass>
        <Restriction>
            <onProperty rdf:resource="&My_University_Ontology;Offers"/>
            <someValuesFrom rdf:resource="&My_University_Ontology;Program"/>
        </Restriction>
    </equivalentClass>
    <rdfs:subClassOf rdf:resource="&My_University_Ontology;Faculty"/>
    <synonymes xml:lang="en">Academic-Department</synonymes>
</Class>

<Class rdf:about="&My_University_Ontology;DiplomaStudent">
    <rdfs:subClassOf rdf:resource="&My_University_Ontology;UnderGraduateStudent"/>
</Class>

<Class rdf:about="&My_University_Ontology;DoctorateStudent">
    <rdfs:subClassOf rdf:resource="&My_University_Ontology;PostGraduateStudent"/>
    <synonymes xml:lang="en">PhD-Student</synonymes>
</Class>

<Class rdf:about="&My_University_Ontology;Event">
    <rdfs:subClassOf rdf:resource="&My_University_Ontology;University"/>
    <synonymes xml:lang="en">news, event-news</synonymes>
</Class>

<Class rdf:about="&My_University_Ontology;Faculty">
    <rdfs:subClassOf rdf:resource="&My_University_Ontology;University"/>
    <synonymes xml:lang="en">Faculty, School, Academic-Department</synonymes>
</Class>

<Class rdf:about="&My_University_Ontology;HeadDepartment">
    <equivalentClass>
        <Restriction>
            <onProperty rdf:resource="&My_University_Ontology;HeadOf"/>
            <onClass rdf:resource="&My_University_Ontology;Department"/>
            <maxQualifiedCardinality rdf:datatype="&xsd;nonNegativeInteger">1
            </maxQualifiedCardinality>
        </Restriction>
```

```
    </equivalentClass>
    <rdfs:subClassOf rdf:resource="&My_University_Ontology;AdministrativeStaff"/>
</Class>

<Class rdf:about="&My_University_Ontology;HeadLibrary">
    <equivalentClass>
        <Restriction>
            <onProperty rdf:resource="&My_University_Ontology;HeadOf"/>
            <someValuesFrom rdf:resource="&My_University_Ontology;Library"/>
        </Restriction>
    </equivalentClass>
    <rdfs:subClassOf rdf:resource="&My_University_Ontology;Library"/>
    <synonymes xml:lang="en">Manager of Library</synonymes>
</Class>

<Class rdf:about="&My_University_Ontology;Laboratory">
    <equivalentClass>
        <Restriction>
            <onProperty rdf:resource="&My_University_Ontology;Contains"/>
            <onClass rdf:resource="&My_University_Ontology;ResearchGroup"/>
            <minQualifiedCardinality rdf:datatype="&xsd;nonNegativeInteger">1
            </minQualifiedCardinality>
        </Restriction>
    </equivalentClass>
    <rdfs:subClassOf rdf:resource="&My_University_Ontology;Faculty"/>
    <synonymes xml:lang="en">Lab</synonymes>
</Class>

<Class rdf:about="&My_University_Ontology;Lecturer">
    <rdfs:subClassOf rdf:resource="&My_University_Ontology;AcademicStaff"/>
    <synonymes xml:lang="en">instructor, trainer, educator</synonymes>
    <hasKey rdf:parseType="Collection">
        <rdf:Description rdf:about="&My_University_Ontology;Teach"/>
    </hasKey>
</Class>

<Class rdf:about="&My_University_Ontology;Library">
    <rdfs:subClassOf rdf:resource="&My_University_Ontology;University"/>
</Class>

<Class rdf:about="&My_University_Ontology;MasterStudent">
    <rdfs:subClassOf rdf:resource="&My_University_Ontology;PostGraduateStudent"/>
    <synonymes xml:lang="en">Master-student</synonymes>
</Class>

<Class rdf:about="&My_University_Ontology;OfficialAssistant">
    <rdfs:subClassOf rdf:resource="&My_University_Ontology;AdministrativeStaff"/>
</Class>

<Class rdf:about="&My_University_Ontology;Person">
    <rdfs:subClassOf rdf:resource="&My_University_Ontology;University"/>
</Class>

<Class rdf:about="&My_University_Ontology;PostDocStudent">
    <rdfs:subClassOf rdf:resource="&My_University_Ontology;Student"/>
    <synonymes xml:lang="en">Post-doc-student</synonymes>
</Class>

<Class rdf:about="&My_University_Ontology;PostGraduateProgram">
    <rdfs:subClassOf rdf:resource="&My_University_Ontology;Program"/>
    <synonymes xml:lang="en">Graduate-Program</synonymes>
</Class>

<Class rdf:about="&My_University_Ontology;PostGraduateStudent">
    <rdfs:subClassOf rdf:resource="&My_University_Ontology;Student"/>
    <synonymes xml:lang="en">Graduate-Student</synonymes>
</Class>

<Class rdf:about="&My_University_Ontology;PostdoctoralProgram">
    <rdfs:subClassOf rdf:resource="&My_University_Ontology;Program"/>
    <synonymes xml:lang="en">Postdoctoral-position, Postdoctoral-fellowship</synonymes>
</Class>

<Class rdf:about="&My_University_Ontology;Professor">
    <rdfs:subClassOf rdf:resource="&My_University_Ontology;AcademicStaff"/>
    <synonymes xml:lang="en">Prof</synonymes>
```

```
    <hasKey rdf:parseType="Collection">
        <rdf:Description rdf:about="&My_University_Ontology;Teach"/>
    </hasKey>
</Class>

<Class rdf:about="&My_University_Ontology;Program">
    <equivalentClass>
        <Restriction>
            <onProperty rdf:resource="&My_University_Ontology;ListOfCourse"/>
            <someValuesFrom rdf:resource="&My_University_Ontology;Course"/>
        </Restriction>
    </equivalentClass>
    <rdfs:subClassOf rdf:resource="&My_University_Ontology;Department"/>
    <synonymes xml:lang="en">Degree-Program, Academic-program, Educational-Degree</synonymes>
    <hasKey rdf:parseType="Collection">
        <rdf:Description rdf:about="&My_University_Ontology;ListOfCourse"/>
    </hasKey>
</Class>

<Class rdf:about="&My_University_Ontology;Project">
    <rdfs:subClassOf rdf:resource="&My_University_Ontology;University"/>
    <synonymes xml:lang="en">Plan, Operation, Idea, Design</synonymes>
</Class>

<Class rdf:about="&My_University_Ontology;Publication">
    <equivalentClass>
        <Restriction>
            <onProperty rdf:resource="&My_University_Ontology;PublishedBy"/>
            <someValuesFrom rdf:resource="&My_University_Ontology;Researcher"/>
        </Restriction>
    </equivalentClass>
    <rdfs:subClassOf rdf:resource="&My_University_Ontology;University"/>
    <synonymes xml:lang="en">Research-publication</synonymes>
</Class>

<Class rdf:about="&My_University_Ontology;ResearchGroup">
    <equivalentClass>
        <Restriction>
            <onProperty rdf:resource="&My_University_Ontology;WorkIn"/>
            <onClass rdf:resource="&My_University_Ontology;Project"/>
            <minQualifiedCardinality rdf:datatype="&xsd;nonNegativeInteger">1
            </minQualifiedCardinality>
        </Restriction>
    </equivalentClass>
    <rdfs:subClassOf rdf:resource="&My_University_Ontology;Laboratory"/>
    <rdfs:subClassOf>
        <Restriction>
            <onProperty rdf:resource="&My_University_Ontology;ComposedOf"/>
            <onClass rdf:resource="&My_University_Ontology;Researcher"/>
            <minQualifiedCardinality rdf:datatype="&xsd;nonNegativeInteger">3
            </minQualifiedCardinality>
        </Restriction>
    </rdfs:subClassOf>
    <rdfs:subClassOf>
        <Restriction>
            <onProperty rdf:resource="&My_University_Ontology;ComposedOf"/>
            <onClass rdf:resource="&My_University_Ontology;Researcher"/>
            <maxQualifiedCardinality rdf:datatype="&xsd;nonNegativeInteger">20
            </maxQualifiedCardinality>
        </Restriction>
    </rdfs:subClassOf>
    <synonymes xml:lang="en">Research-team</synonymes>
    <hasKey rdf:parseType="Collection">
        <rdf:Description rdf:about="&My_University_Ontology;ResearcherFiled"/>
    </hasKey>
</Class>

<Class rdf:about="&My_University_Ontology;Researcher">
    <equivalentClass>
        <Restriction>
            <onProperty rdf:resource="&My_University_Ontology;BelongsTo"/>
            <someValuesFrom rdf:resource="&My_University_Ontology;ResearchGroup"/>
        </Restriction>
    </equivalentClass>
    <rdfs:subClassOf rdf:resource="&My_University_Ontology;AcademicStaff"/>
    <synonymes xml:lang="en">research-worker, scientist</synonymes>
```

```
        <hasKey rdf:parseType="Collection">
            <rdf:Description rdf:about="&My_University_Ontology;ResearcherFiled"/>
        </hasKey>
    </Class>

    <Class rdf:about="&My_University_Ontology;Staff">
        <equivalentClass>
            <Restriction>
                <onProperty rdf:resource="&My_University_Ontology;WorkIn"/>
                <someValuesFrom rdf:resource="&My_University_Ontology;Department"/>
            </Restriction>
        </equivalentClass>
        <rdfs:subClassOf rdf:resource="&My_University_Ontology;Person"/>
        <synonymes xml:lang="en">employee, worker, people, educational-employee</synonymes>
    </Class>

    <Class rdf:about="&My_University_Ontology;Student">
        <equivalentClass>
            <Restriction>
                <onProperty rdf:resource="&My_University_Ontology;StudyIn"/>
                <onClass rdf:resource="&My_University_Ontology;Program"/>
                <minQualifiedCardinality rdf:datatype="&xsd;nonNegativeInteger">1
                </minQualifiedCardinality>
            </Restriction>
        </equivalentClass>
        <equivalentClass>
            <Restriction>
                <onProperty rdf:resource="&My_University_Ontology;EnrolledBy"/>
                <someValuesFrom rdf:resource="&My_University_Ontology;University"/>
            </Restriction>
        </equivalentClass>
        <rdfs:subClassOf rdf:resource="&My_University_Ontology;Person"/>
        <synonymes xml:lang="en">current-student, prospective student</synonymes>
    </Class>

    <Class rdf:about="&My_University_Ontology;Thesis">
        <rdfs:subClassOf rdf:resource="&My_University_Ontology;Publication"/>
        <synonymes xml:lang="en">Dissertation, Theory</synonymes>
        <hasKey rdf:parseType="Collection">
            <rdf:Description rdf:about="&My_University_Ontology;ThesisAuthor"/>
        </hasKey>
    </Class>

    <Class rdf:about="&My_University_Ontology;UnderGraduateProgram">
        <rdfs:subClassOf rdf:resource="&My_University_Ontology;Program"/>
        <synonymes xml:lang="en">First tertiary degree</synonymes>
    </Class>

    <Class rdf:about="&My_University_Ontology;UnderGraduateStudent">
        <rdfs:subClassOf rdf:resource="&My_University_Ontology;Student"/>
    </Class>

    <Class rdf:about="&My_University_Ontology;University">
        <synonymes xml:lang="en">University college</synonymes>
    </Class>

    <Class rdf:about="&My_University_Ontology;UniversityDean">
        <equivalentClass>
            <Restriction>
                <onProperty rdf:resource="&My_University_Ontology;HeadOf"/>
                <onClass rdf:resource="&My_University_Ontology;University"/>
                <maxQualifiedCardinality rdf:datatype="&xsd;nonNegativeInteger">1
                </maxQualifiedCardinality>
            </Restriction>
        </equivalentClass>
        <rdfs:subClassOf rdf:resource="&My_University_Ontology;AdministrativeStaff"/>
        <synonymes xml:lang="en">Manager, Chair</synonymes>
    </Class>


</rdf:RDF>
```

Annexe B

Scripts DDL de création des sources de données.

Cette annexe contient les scripts DDL (*Data Definition language*) en SQL des deux sources de données présenté dans le chapitre 4 :

1- Source de données 1 (SGBD Microsoft SQL Server) :

```
USE [MASTER]
GO
/****** OBJECT:  DATABASE [DATASOURCE1]    SCRIPT DATE: 12/05/2014 10:03:11 ******/
CREATE DATABASE [DATASOURCE1]
 CONTAINMENT = NONE ON  PRIMARY
( NAME = N'DATASOURCE1', FILENAME = N'C:\PROGRAM FILES\MICROSOFT SQL
SERVER\MSSQL10.SQLEXPRESS\MSSQL\DATA\DATASOURCE1.MDF' , SIZE = 5120KB , MAXSIZE = UNLIMITED,
FILEGROWTH = 1024KB )
 LOG ON
( NAME = N'DATASOURCE1_LOG', FILENAME = N'C:\PROGRAM FILES\MICROSOFT SQL
SERVER\MSSQL10.SQLEXPRESS\MSSQL\DATA\DATASOURCE1_LOG.LDF' , SIZE = 2048KB , MAXSIZE = 2048GB ,
FILEGROWTH = 10%)
GO

ALTER DATABASE [DATASOURCE1] SET COMPATIBILITY_LEVEL = 110
GO

USE [DATASOURCE1]
GO
/****** OBJECT:  TABLE [DBO].[COURSES]  ******/
SET ANSI_NULLS ON
GO
SET QUOTED_IDENTIFIER ON
GO
SET ANSI_PADDING ON
GO
CREATE TABLE [DBO].[COURSES](
        [COURSE_ID] [INT] NOT NULL,
        [TEACHER_ID] [INT] NOT NULL,
        [PROGRAM_ID] [INT] NOT NULL,
        [NAME] [VARCHAR](45) NULL,
 CONSTRAINT [PK_COURSE] PRIMARY KEY CLUSTERED
(
        [COURSE_ID] ASC
)WITH (PAD_INDEX = OFF, STATISTICS_NORECOMPUTE = OFF, IGNORE_DUP_KEY = OFF, ALLOW_ROW_LOCKS = ON,
ALLOW_PAGE_LOCKS = ON) ON [PRIMARY]
) ON [PRIMARY]

GO
SET ANSI_PADDING OFF
GO
/****** OBJECT:  TABLE [DBO].[PROGRAMS]  ******/
SET ANSI_NULLS ON
GO
SET QUOTED_IDENTIFIER ON
GO
SET ANSI_PADDING ON
GO
CREATE TABLE [DBO].[PROGRAMS](
        [PROGRAM_ID] [INT] NOT NULL,
        [NAME] [VARCHAR](45) NULL,
        [TYPE] [VARCHAR](45) NULL,
 CONSTRAINT [PK_PROGRAM] PRIMARY KEY CLUSTERED
(
        [PROGRAM_ID] ASC
)WITH (PAD_INDEX = OFF, STATISTICS_NORECOMPUTE = OFF, IGNORE_DUP_KEY = OFF, ALLOW_ROW_LOCKS = ON,
ALLOW_PAGE_LOCKS = ON) ON [PRIMARY]
) ON [PRIMARY]

GO
```

```
SET ANSI_PADDING OFF
GO
/****** OBJECT:  TABLE [DBO].[STAFFS]    ******/
SET ANSI_NULLS ON
GO
SET QUOTED_IDENTIFIER ON
GO
SET ANSI_PADDING ON
GO
CREATE TABLE [DBO].[STAFFS](
	[STAFF_ID] [INT] NOT NULL,
	[NAME] [VARCHAR](45) NULL,
 CONSTRAINT [PK_STAFF] PRIMARY KEY CLUSTERED
(
	[STAFF_ID] ASC
)WITH (PAD_INDEX = OFF, STATISTICS_NORECOMPUTE = OFF, IGNORE_DUP_KEY = OFF, ALLOW_ROW_LOCKS = ON,
ALLOW_PAGE_LOCKS = ON) ON [PRIMARY]
) ON [PRIMARY]

GO
SET ANSI_PADDING OFF
GO
/****** OBJECT:  TABLE [DBO].[STUDENTS]    ******/
SET ANSI_NULLS ON
GO
SET QUOTED_IDENTIFIER ON
GO
SET ANSI_PADDING ON
GO
CREATE TABLE [DBO].[STUDENTS](
	[STUDENT_ID] [INT] NOT NULL,
	[PROGRAM_ID] [INT] NOT NULL,
	[NAME] [VARCHAR](45) NULL,
 CONSTRAINT [PK_STUDENT] PRIMARY KEY CLUSTERED
(
	[STUDENT_ID] ASC
)WITH (PAD_INDEX = OFF, STATISTICS_NORECOMPUTE = OFF, IGNORE_DUP_KEY = OFF, ALLOW_ROW_LOCKS = ON,
ALLOW_PAGE_LOCKS = ON) ON [PRIMARY]
) ON [PRIMARY]

GO
SET ANSI_PADDING OFF
GO
/****** OBJECT:  TABLE [DBO].[TEACHERS]    ******/
SET ANSI_NULLS ON
GO
SET QUOTED_IDENTIFIER ON
GO
SET ANSI_PADDING ON
GO
CREATE TABLE [DBO].[TEACHERS](
	[TEACHER_ID] [INT] NOT NULL,
	[LEVEL_CODE] [VARCHAR](5) NULL,
	[NAME] [VARCHAR](45) NULL,
 CONSTRAINT [PK_TEACHER] PRIMARY KEY CLUSTERED
(
	[TEACHER_ID] ASC
)WITH (PAD_INDEX = OFF, STATISTICS_NORECOMPUTE = OFF, IGNORE_DUP_KEY = OFF, ALLOW_ROW_LOCKS = ON,
ALLOW_PAGE_LOCKS = ON) ON [PRIMARY]
) ON [PRIMARY]

GO
SET ANSI_PADDING OFF
GO
ALTER TABLE [DBO].[STUDENTS]  WITH CHECK ADD  CONSTRAINT [FK_STUDENT_PROGRAM] FOREIGN
KEY([PROGRAM_ID])
REFERENCES [DBO].[PROGRAMS] ([PROGRAM_ID])
GO
ALTER TABLE [DBO].[STUDENTS] CHECK CONSTRAINT [FK_STUDENT_PROGRAM]
GO
```

2- Source de données 2 (SGBD Oracle Database) :

```
--------------------------------------------------------
--  Fichier créé - lundi-mai-12-2014
--------------------------------------------------------
--  DDL for Table SCHEMA
--------------------------------------------------------

CREATE USER DATASOURCE2
IDENTIFIED BY <password>
DEFAULT TABLESPACE USERS
TEMPORARY TABLESPACE TEMP
PROFILE DEFAULT
ACCOUNT UNLOCK;

-- 3 Roles for DATASOURCE2
GRANT DBA TO DATASOURCE2;
GRANT CONNECT TO DATASOURCE2;
GRANT RESOURCE TO DATASOURCE2;
ALTER USER DATASOURCE2 DEFAULT ROLE ALL;

-- 8 System Privileges for DATASOURCE2
GRANT CREATE DATABASE LINK TO DATASOURCE2;
GRANT ALTER SESSION TO DATASOURCE2;
GRANT CREATE SESSION TO DATASOURCE2;
GRANT CREATE TABLE TO DATASOURCE2;
GRANT UNLIMITED TABLESPACE TO DATASOURCE2;
GRANT CREATE SEQUENCE TO DATASOURCE2;
GRANT CREATE VIEW TO DATASOURCE2;
GRANT CREATE SYNONYM TO DATASOURCE2;

-- 1 Tablespace Quota for DATASOURCE2
ALTER USER DATASOURCE2 QUOTA UNLIMITED ON USERS;
--------------------------------------------------------
--  DDL for TABLES
--------------------------------------------------------
CREATE TABLE DATASOURCE2.COURSES
(
  COURSE_ID      INTEGER                        NOT NULL,
  NAME           VARCHAR2(45 CHAR),
  DPTE_CODE      INTEGER,
  INSTRUCTOR_ID  INTEGER
)

CREATE TABLE DATASOURCE2.DEPARTEMENTS
(
  DPTE_CODE  INTEGER                            NOT NULL,
  NAME       VARCHAR2(45 CHAR)
)

CREATE TABLE DATASOURCE2.EMPLOYEES
(
  EMPLOYEE_ID  INTEGER                          NOT NULL,
  NAME         VARCHAR2(45 CHAR),
  DPTE_CODE    INTEGER
)

CREATE TABLE DATASOURCE2.INSTRUCTORS
(
  INSTRUCTOR_ID  INTEGER                        NOT NULL,
  NAME           VARCHAR2(45 CHAR)
)

CREATE TABLE DATASOURCE2.STUDENTS
(
  STUDENT_ID  INTEGER                           NOT NULL,
  NAME        VARCHAR2(45 BYTE),
  DPTE_CODE   INTEGER
)
```

```
-----------------------------------------------------
--  DDL for INDEXS
-----------------------------------------------------
CREATE UNIQUE INDEX DATASOURCE2.DEPARTEMENT_PK ON DATASOURCE2.DEPARTEMENTS
(DPTE CODE)
LOGGING
TABLESPACE USERS
PCTFREE    10
INITRANS   2
MAXTRANS   255

CREATE UNIQUE INDEX DATASOURCE2.EMPLOYEE_PK ON DATASOURCE2.EMPLOYEES
(EMPLOYEE_ID)
LOGGING
TABLESPACE USERS
PCTFREE    10
INITRANS   2
MAXTRANS   255

CREATE UNIQUE INDEX DATASOURCE2.INSTRUCTOR_PK ON DATASOURCE2.INSTRUCTORS
(INSTRUCTOR_ID)
LOGGING
TABLESPACE USERS
PCTFREE    10
INITRANS   2
MAXTRANS   255

CREATE UNIQUE INDEX DATASOURCE2.STUDENT_PK ON DATASOURCE2.STUDENTS
(STUDENT_ID)
LOGGING
TABLESPACE USERS
PCTFREE    10
INITRANS   2
MAXTRANS   255

CREATE UNIQUE INDEX DATASOURCE2.COURSE_PK ON DATASOURCE2.COURSES
(COURSE_ID)
LOGGING
TABLESPACE USERS
PCTFREE    10
INITRANS   2
MAXTRANS   255
STORAGE    (
            INITIAL          64K
            NEXT             1M
            MINEXTENTS       1
            MAXEXTENTS       UNLIMITED
            PCTINCREASE      0
            BUFFER_POOL      DEFAULT
           )

-----------------------------------------------------
--  DDL for CONSTRAINTS PRIMARY KEYS
-----------------------------------------------------
ALTER TABLE DATASOURCE2.DEPARTEMENTS ADD (
  CONSTRAINT DEPARTEMENT_PK
 PRIMARY KEY
 (DPTE CODE)
    USING INDEX
    TABLESPACE USERS
    PCTFREE    10
    INITRANS   2
    MAXTRANS   255
);

ALTER TABLE DATASOURCE2.EMPLOYEES ADD (
  CONSTRAINT EMPLOYEE_PK
 PRIMARY KEY
 (EMPLOYEE ID)
    USING INDEX
    TABLESPACE USERS
    PCTFREE    10
    INITRANS   2
    MAXTRANS   255
);
```

```
ALTER TABLE DATASOURCE2.INSTRUCTORS ADD (
  CONSTRAINT INSTRUCTOR_PK
 PRIMARY KEY
 (INSTRUCTOR ID)
    USING INDEX
    TABLESPACE USERS
    PCTFREE    10
    INITRANS   2
    MAXTRANS   255
);

ALTER TABLE DATASOURCE2.STUDENTS ADD (
  CONSTRAINT STUDENT_PK
 PRIMARY KEY
 (STUDENT ID)
    USING INDEX
    TABLESPACE USERS
    PCTFREE    10
    INITRANS   2
    MAXTRANS   255
);

ALTER TABLE DATASOURCE2.COURSES ADD (
  CONSTRAINT COURSE_PK
 PRIMARY KEY
 (COURSE ID)
    USING INDEX
    TABLESPACE USERS
    PCTFREE    10
    INITRANS   2
    MAXTRANS   255
);

--------------------------------------------------------
--  DDL for CONSTRAINTS FOREIGN KEYS
--------------------------------------------------------

ALTER TABLE DATASOURCE2.EMPLOYEES ADD (
  CONSTRAINT FK_EMPLOYEE_DPTE
 FOREIGN KEY (DPTE_CODE)
 REFERENCES DATASOURCE2.DEPARTEMENTS (DPTE_CODE));

ALTER TABLE DATASOURCE2.STUDENTS ADD (
  CONSTRAINT FK_STUDENT_DPTE
 FOREIGN KEY (DPTE_CODE)
 REFERENCES DATASOURCE2.DEPARTEMENTS (DPTE_CODE));

ALTER TABLE DATASOURCE2.COURSES ADD (
  CONSTRAINT FK_COURSE_INSTRUCTOR
 FOREIGN KEY (INSTRUCTOR_ID)
 REFERENCES DATASOURCE2.INSTRUCTORS (INSTRUCTOR_ID),
  CONSTRAINT FK_COURSE_DPTE
 FOREIGN KEY (DPTE_CODE)
 REFERENCES DATASOURCE2.DEPARTEMENTS (DPTE_CODE))
```

Annexe C

Code ASP.Net C# d'implémentation de l'interface HERO-Med.

Cette annexe contient quel que code ASP.Net et C# de l'implémentation de l'interface graphique du Prototype HERO-Med :

```
------------------------------------------------------
--  ASP.NET HTML CODE FOR MASTER PAGE
------------------------------------------------------
<%@ Master Language="C#"AutoEventWireup="true" CodeFile="Site.master.cs"Inherits="Site"%>

<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
"http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html xmlns="http://www.w3.org/1999/xhtml">
<head runat="server">
    <title>Higher Education Reference Ontology Mediator</title>
    <meta name="robots" content="noindex, nofollow" />
    <link type="text/css" rel="Stylesheet" href="Content/styles.css" />
    <link type="text/css" rel="Stylesheet" href="Content/sprite.css" />
    <script type="text/javascript" src="Scripts/Hero.js"></script>
</head>
<body>
    <form id="form1" runat="server">
        <dx:ASPxSplitter ID="LayoutSplitter" runat="server" FullscreenMode="True"
Width="100%" Height="100%" Orientation="Vertical" SaveStateToCookies="True"
ResizingMode="Live">
            <Panes>
                <dx:SplitterPane Size="80" Name="HeaderPane">
                    <ContentCollection>
                        <dx:SplitterContentControl runat="server" CssClass="HeadPane">
                            <table id="HeaderArea">
                                <tr>
                                    <td>
                                        <dx:ASPxImage ID="ASPxImage1" runat="server"
Height="49px" ImageUrl="~/Content/system_icon.png" ShowLoadingImage="True" Width="48px">
                                        </dx:ASPxImage>
                                    </td>
                                    <td class="auto-style7"> <span class="auto-
style5"><strong>HERO</strong></span>
                                        <strong><span class="auto-style4"><span
class="style4">Mediator</span></span><span><span class="auto-
style6">®</span></span></strong></td>
                                    <td class="auto-style2" style="font-weight: 700;">
                                        <dx:ASPxMenu ID="InfoMenu" runat="server"
ClientInstanceName="ClientInfoMenu"
                                            CssClass="InfoMenu"
DataSourceID="InfoMenuDataSource"
                                            OnItemDataBound="InfoMenu_OnItemDataBound"
RenderMode="Lightweight"
                                            SeparatorWidth="0px" ShowAsToolbar="True">
                                            <ClientSideEvents
ItemClick="HeroDemo.ClientInfoMenu_ItemClick" />
                                            <SubMenuStyle CssClass="SubMenu" />
                                            <Border BorderWidth="0px" />
                                        </dx:ASPxMenu>
```

```
                                        <asp:XmlDataSource ID="InfoMenuDataSource"
runat="server" DataFile="~/App_Data/InfoLayout.xml"
XPath="Items/Item"></asp:XmlDataSource>
                                    </td>
                                    <td id="SearchBoxSpacer" class="Spacer"
runat="server"><b></b>
                                    </td>
                                    <td>
                                        <dx:ASPxButtonEdit runat="server" ID="SearchBox"
Width="220" Height="31" NullText="Type to Search..." CssClass="SearchBox"
                                            ClientInstanceName="ClientSearchBox" Font-
Size="12px">
                                            <ClientSideEvents

TextChanged="MailDemo.ClientSearchBox_TextChanged"

KeyDown="MailDemo.ClientSearchBox_KeyDown"

KeyPress="MailDemo.ClientSearchBox_KeyPress" />
                                            <Buttons>
                                                <dx:EditButton>
                                                </dx:EditButton>
                                            </Buttons>
                                            <ButtonStyle CssClass="SearchBoxButton" />
                                            <NullTextStyle Font-Italic="true" />
                                        </dx:ASPxButtonEdit>
                                    </td>
                                </tr>
                            </table>
                            <dx:ASPxMenu runat="server" ID="HeaderMenu" Width="100%"
ItemAutoWidth="False" OnItemClick="HeaderMenu_ItemClick" AllowSelectItem="True">
                                <Items>
                                    <dx:MenuItem Text="Navigate">
                                        <Items>
                                            <dx:MenuItem NavigateUrl="~/Default.aspx"
Text="Ontology">

                                                <Image IconID="chart_bubble_16x16">
                                                </Image>
                                            </dx:MenuItem>
                                            <dx:MenuItem NavigateUrl="~/FormData.aspx"
Text="Data Sources">

                                                <Image IconID="data_database_16x16">
                                                </Image>
                                            </dx:MenuItem>
                                            <dx:MenuItem NavigateUrl="~/FormMapping.aspx"
Text="Mapping info">

                                                <Image

IconID="chart_fullstackedline_16x16">

                                                </Image>
                                            </dx:MenuItem>
                                            <dx:MenuItem
NavigateUrl="~/FormReporting.aspx" Text="Reporting">
                                                <Image IconID="chart_stackedbar_16x16">
                                                </Image>
                                            </dx:MenuItem>
                                            <dx:MenuItem Text="Print..."
                                                <Image IconID="print_print_16x16">
                                                </Image>
                                            </dx:MenuItem>
                                            <dx:MenuItem Text="Exit"
                                                <Image IconID="actions_cancel_16x16">
                                                </Image>
```

```
                                        </dx:MenuItem>
                                    </Items>
                                    <Image IconID="navigation_navigationbar_16x16">
                                    </Image>
                                </dx:MenuItem>
                                <dx:MenuItem Text="Edit">
                                    <Items>
                                        <dx:MenuItem Text="Cut"
                                            <Image IconID="edit_cut_16x16">
                                            </Image>
                                        </dx:MenuItem>
                                        <dx:MenuItem Text="Copy"
                                            <Image IconID="edit_copy_16x16">
                                            </Image>
                                        </dx:MenuItem>
                                        <dx:MenuItem Text="Paste"
                                            <Image IconID="edit_paste_16x16">
                                            </Image>
                                        </dx:MenuItem>
                                        <dx:MenuItem Text="Delete" BeginGroup="True"
                                            <Image IconID="edit_delete_16x16">
                                            </Image>
                                        </dx:MenuItem>
                                    </Items>
                                    <Image IconID="tasks_edittask_16x16">
                                    </Image>
                                </dx:MenuItem>
                                <dx:MenuItem Text="Ontology"
NavigateUrl="~/Default.aspx">
                                    <Items>
                                        <dx:MenuItem Text="Expaned">
                                            <Image IconID="actions_stretch_16x16">
                                            </Image>
                                        </dx:MenuItem>
                                        <dx:MenuItem Text="Collapse">
                                            <Image IconID="actions_squeeze_16x16">
                                            </Image>
                                        </dx:MenuItem>
                                        <dx:MenuItem Text="Ontology Graph"
NavigateUrl="~/DefaultGraph.aspx" BeginGroup="True">
                                            <Image IconID="chart_bubble_16x16">
                                            </Image>
                                        </dx:MenuItem>
                                        <dx:MenuItem NavigateUrl="~/DefaultURM.aspx"
Text="Update Ontology URM">
                                            <Image
IconID="chart_chartsshowlegend_16x16">
                                            </Image>
                                        </dx:MenuItem>
                                    </Items>
                                    <Image Url="~/OntoGraph/ontology.gif">
                                    </Image>
                                </dx:MenuItem>
                                <dx:MenuItem Text="Query">
                                    <Items>
                                        <dx:MenuItem Text="Create" Name="Create">
                                            <Image
IconID="miscellaneous_design_16x16">
                                            </Image>
                                        </dx:MenuItem>
                                        <dx:MenuItem Text="Run" Name="Run"
Enabled="False">
```

```
                                        <Image IconID="arrows_next_16x16">
                                        </Image>
                                    </dx:MenuItem>
                                    <dx:MenuItem Text="Clear" Name="Clear"
Enabled="False">
                                        <Image IconID="actions_clear_16x16">
                                        </Image>
                                    </dx:MenuItem>
                                    <dx:MenuItem Text="Save" Name="Save"
Enabled="False" BeginGroup="True">
                                        <Image IconID="save_saveto_16x16">
                                        </Image>
                                    </dx:MenuItem>
                                    <dx:MenuItem Text="Load" Name="Load">
                                        <Image IconID="support_article_16x16">
                                        </Image>
                                    </dx:MenuItem>
                                </Items>
                                <Image IconID="data_sortasc_16x16">
                                </Image>
                            </dx:MenuItem>
                            <dx:MenuItem Text="Sources"
NavigateUrl="~/FormData.aspx">
                                <Items>
                                    <dx:MenuItem Text="Add source">
                                        <Image
IconID="data_addnewdatasource_16x16">
                                        </Image>
                                    </dx:MenuItem>
                                    <dx:MenuItem Text="Delete source">
                                        <Image
IconID="data_deletedatasource_16x16">
                                        </Image>
                                    </dx:MenuItem>
                                </Items>
                                <Image IconID="data_database_16x16">
                                </Image>
                            </dx:MenuItem>
                            <dx:MenuItem Text="Mapping"
NavigateUrl="~/FormMapping.aspx">
                                <Items>
                                    <dx:MenuItem>
                                    </dx:MenuItem>
                                    <dx:MenuItem>
                                    </dx:MenuItem>
                                    <dx:MenuItem>
                                    </dx:MenuItem>
                                </Items>
                                <Image IconID="chart_fullstackedline_16x16">
                                </Image>
                            </dx:MenuItem>
                            <dx:MenuItem Text="Tools"
NavigateUrl="~/FormReporting.aspx">
                                <Items>
                                    <dx:MenuItem>
                                    </dx:MenuItem>
                                    <dx:MenuItem>
                                    </dx:MenuItem>
                                    <dx:MenuItem>
                                    </dx:MenuItem>
                                </Items>
                                <Image IconID="programming_ide_16x16">
```

```
                                    </Image>
                                </dx:MenuItem>
                                <dx:MenuItem Text="Help">
                                    <Items>
                                        <dx:MenuItem Text="About HERO Mediator ®"
                                            <Image IconID="people_customer_16x16">
                                            </Image>
                                        </dx:MenuItem>
                                        <dx:MenuItem Text="Download Free Trial"
                                            <Image IconID="navigation_next_16x16">
                                            </Image>
                                        </dx:MenuItem>
                                        <dx:MenuItem Text="Buy Now"
                    <Image IconID="miscellaneous_currency_16x16">
                                            </Image>
                                        </dx:MenuItem>
                                    </Items>
                                    <Image IconID="support_index_16x16">
                                    </Image>
                                </dx:MenuItem>
                            </Items>
                        </dx:ASPxMenu>
                    </dx:SplitterContentControl>
                </ContentCollection>
                <Separator Visible="True">
                </Separator>
                <PaneStyle CssClass="HeaderPane">
                    <BorderBottom BorderWidth="0" />
                    <Paddings Padding="0" />
                </PaneStyle>
            </dx:SplitterPane>
            <dx:SplitterPane Name="BodyPane">
                <Panes>
                    <dx:SplitterPane ShowCollapseBackwardButton="True" MinSize="150"
                        MaxSize="300px" Size="300">
                        <Panes>
                            <dx:SplitterPane ScrollBars="Auto" Name="SidePane">
                                <ContentCollection>
                                    <dx:SplitterContentControl runat="server">
                                        <asp:ContentPlaceHolder ID="SideHolder"
runat="server" />
                                    </dx:SplitterContentControl>
                                </ContentCollection>
                                <PaneStyle>
                                    <BorderBottom BorderWidth="1" />
                                    <Paddings Padding="0" />
                                </PaneStyle>
                            </dx:SplitterPane>
                            <dx:SplitterPane Name="CornerPane" AllowResize="False"
ShowSeparatorImage="False" Size="165">
                                <PaneStyle>
                                    <Paddings Padding="0" />
                                </PaneStyle>
                                <ContentCollection>
                                    <dx:SplitterContentControl runat="server">
                                        <dx:ASPxMenu runat="server" ID="CornerMenu"
RenderMode="Lightweight" Width="100%" Orientation="Vertical"
                                            ClientInstanceName="ClientCornerMenu"
CssClass="CornerMenu">
                                            <Border BorderWidth="0" />
                                            <Items>
```

```
                                                    <dx:MenuItem Text="Ontology Query
Modele" NavigateUrl="~/Default.aspx">
                                                        <Image
IconID="chart_bubble_32x32"></Image>
                                                    </dx:MenuItem>
                                                    <dx:MenuItem Text="Data Sources
Schemas" NavigateUrl="~/FormData.aspx">
                                                        <Image
IconID="data_addnewdatasource_32x32"></Image>
                                                    </dx:MenuItem>
                                                    <dx:MenuItem Text="Mapping
information" NavigateUrl="~/FormMapping.aspx">
                                                        <Image
IconID="chart_stackedline_32x32"></Image>
                                                    </dx:MenuItem>
                                                    <dx:MenuItem Text="Reporting"
NavigateUrl="~/FormReporting.aspx">
                                                        <Image
IconID="chart_bar_32x32"></Image>
                                                    </dx:MenuItem>
                                                </Items>
                                            </dx:ASPxMenu>
                                        </dx:SplitterContentControl>
                                    </ContentCollection>
                                </dx:SplitterPane>
                            </Panes>
                            <ContentCollection>
                                <dx:SplitterContentControl
runat="server"></dx:SplitterContentControl>
                            </ContentCollection>
                        </dx:SplitterPane>
                        <dx:SplitterPane Name="MainPane" AllowResize="True">
                            <ContentCollection>
                                <dx:SplitterContentControl runat="server">
                                    <asp:ContentPlaceHolder ID="MainHolder"
runat="server" />
                                </dx:SplitterContentControl>
                            </ContentCollection>
                            <PaneStyle>
                                <Border BorderWidth="0" />
                                <Paddings Padding="0" />
                            </PaneStyle>
                        </dx:SplitterPane>
                    </Panes>
                    <Separator Visible="True" />
                    <PaneStyle>
                        <BorderBottom BorderWidth="0" />
                    </PaneStyle>
                    <ContentCollection>
                        <dx:SplitterContentControl
runat="server"></dx:SplitterContentControl>
                    </ContentCollection>
                </dx:SplitterPane>
                <dx:SplitterPane Name="FooterPane" ShowSeparatorImage="True">
                    <Separator Visible="True">
                    </Separator>
                    <ContentCollection>
                        <dx:SplitterContentControl runat="server">
                            <table class="style1">
                                <tr>
                                    <td class="style15">
                                        <dx:ASPxLabel ID="ASPxLabel2" runat="server"
```

```
                                        Style="text-align: left; font-weight: 700">
                                    </dx:ASPxLabel>
                                </td>
                                <td class="style17">All Rigths Reserved Samorcool
Corporation 2014</td>
                                <td class="style21">
                                    <dx:ASPxMenu ID="ASPxMenu4" runat="server"
DataSourceID="XmlDataSourceFooter"
                                        EnableHotTrack="False"
RenderMode="Lightweight"
                                        Style="margin-left: 0px; margin-right: 0px;
text-align: right;" Width="100%">
                                        <Paddings Padding="0px" />
                                        <Border BorderWidth="0px" />
                                    </dx:ASPxMenu>
                                </td>
                            </tr>
                            <tr>
                                <td class="style18"> </td>
                                <td class="style19"></td>
                                <td class="style20"></td>
                            </tr>
                        </table>
                    </dx:SplitterContentControl>
                </ContentCollection>
            </dx:SplitterPane>
        </Panes>
        <ClientSideEvents Init="HeroDemo.ClientLayoutSplitter_Init" />
    </dx:ASPxSplitter>
    <asp:XmlDataSource ID="XmlDataSourceHeader" runat="server"
        DataFile="~/App_Data/TopMenu.xml" XPath="/items/*"></asp:XmlDataSource>
    <asp:XmlDataSource ID="XmlDataSourceFooter" runat="server"
        DataFile="~/App_Data/TopMenu.xml" XPath="/items/*"></asp:XmlDataSource>
</form>
</body>
</html>

------------------------------------------------------
--  C# CODE BEHIND FILE FOR MASTER PAGE
------------------------------------------------------
using System;
using System.Web.UI;
using System.Xml;
using DevExpress.Utils;
using DevExpress.Web.ASPxMenu;
using DevExpress.Web.ASPxEditors;
using DevExpress.Web.ASPxSplitter;
using System.Runtime.InteropServices;

public partial class Site : MasterPage {
    protected void Page_Load(object sender, EventArgs e) {
        HeaderMenu.ShowPopOutImages = DevExpress.Utils.DefaultBoolean.False;
        ASPxLabel2.Text = DateTime.Now.Year + Server.HtmlDecode(" &copy; Copyright by
[Samir SELLAMI]");
        InfoMenu.ShowPopOutImages = DefaultBoolean.False;
    }
    protected void InfoMenu_OnItemDataBound(object sender, MenuItemEventArgs e)
    {
        IHierarchyData itemHierarchyData = (IHierarchyData)e.Item.DataItem;
        var element = (XmlElement)itemHierarchyData.Item;
        var classAttr = element.Attributes["SpriteClassName"];
        if (classAttr != null)
            e.Item.Image.SpriteProperties.CssClass = classAttr.Value;
```

```
        if (e.Item.Parent.Name == "theme" && e.Item.Name == Utils.CurrentTheme)
            e.Item.Selected = true;
        if (e.Item.Name == "print")
        {
            var url = GetPrintItemNavigationUrl();
            if (string.IsNullOrEmpty(url))
                e.Item.Enabled = false;
            else
                e.Item.NavigateUrl = url;
        }
    }
    protected string GetPrintItemNavigationUrl()
    {
        switch (Utils.CurrentPageName)
        {
            case "mail":
                return "../PrintDefault.aspx";
            case "data":
                return "../PrintFormData.aspx";
            case "mapping":
                return "../PrintFormMapping.aspx";
            case "graph":
                return "../PrintDefaultGraph.aspx";
            case "urm":
                return "../PrintDefaultURM.aspx";
        }
        return string.Empty;
    }
    protected void HeaderMenu_ItemClick(object source, MenuItemEventArgs e)
    {
        ////////////////////// File Menu ////////////////////////////////////
        if (e.Item.Text == "Exit")
        {
             Application.Close();
        }

        if (e.Item.Text == "Print...")
        {
            var url1 = GetPrintItemNavigationUrl();
            if (string.IsNullOrEmpty(url1))
                e.Item.Enabled = false;
            else
                e.Item.NavigateUrl = url1;
        }

        ////////////////////// Edit Menu ////////////////////////////////////
        if (e.Item.Text == "Copy")
        {
            keybd_event(VK_LCONTROL, 0, KEYEVENTF_EXTENDEDKEY, 0);
            keybd_event(C, 0, KEYEVENTF_EXTENDEDKEY, 0);
            keybd_event(C, 0, KEYEVENTF_KEYUP, 0);
            keybd_event(VK_LCONTROL, 0, KEYEVENTF_KEYUP, 0);
        }
        if (e.Item.Text == "Cut")
        {
            keybd_event(VK_LCONTROL, 0, KEYEVENTF_EXTENDEDKEY, 0);
            keybd_event(X, 0, KEYEVENTF_EXTENDEDKEY, 0);
            keybd_event(X, 0, KEYEVENTF_KEYUP, 0);
            keybd_event(VK_LCONTROL, 0, KEYEVENTF_KEYUP, 0);
        }
        if (e.Item.Text == "Paste")
        {
```

```
            keybd_event(VK_LCONTROL, 0, KEYEVENTF_EXTENDEDKEY, 0);
            keybd_event(V, 0, KEYEVENTF_EXTENDEDKEY, 0);
            keybd_event(V, 0, KEYEVENTF_KEYUP, 0);
            keybd_event(VK_LCONTROL, 0, KEYEVENTF_KEYUP, 0);

        }
        ////////////////////// Ontology Menu ////////////////////////////////////
        if (e.Item.Text == "Expaned")
        {
            ASPxButton O1 = (ASPxButton)SideHolder.FindControl("btnExpandAll");
            if (O1 != null)
            {
                ((IPostBackEventHandler)O1).RaisePostBackEvent(null);
            }
        }
        if (e.Item.Text == "Collapse")
        {
            ASPxButton O2 = (ASPxButton)SideHolder.FindControl("btnCollapseAll");
            if (O2 != null)
            {
                ((IPostBackEventHandler)O2).RaisePostBackEvent(null);
            }
        }
        ////////////////////// Query Menu ////////////////////////////////////
        ASPxSplitter SP = (ASPxSplitter)MainHolder.FindControl("ASPxSplitter1");

        if (e.Item.Text == "Create")
        {
            HeaderMenu.Items.FindByText("Run").Enabled =
HeaderMenu.Items.FindByText("Clear").Enabled =
HeaderMenu.Items.FindByText("Save").Enabled = true;
            ASPxButton C1 = (ASPxButton)SP.FindControl("btnCreate");
            if (C1 != null)
            {
                ((IPostBackEventHandler)C1).RaisePostBackEvent(null);
            }
        }
        if (e.Item.Text == "Clear")
        {
            HeaderMenu.Items.FindByText("Run").Enabled =
HeaderMenu.Items.FindByText("Clear").Enabled =
HeaderMenu.Items.FindByText("Save").Enabled = false;
            ASPxButton C2 = (ASPxButton)SP.FindControl("btnClear");
            if (C2 != null)
            {
                ((IPostBackEventHandler)C2).RaisePostBackEvent(null);
            }
        }
        if (e.Item.Text == "Save")
        {
            HeaderMenu.Items.FindByText("Run").Enabled =
HeaderMenu.Items.FindByText("Clear").Enabled =
HeaderMenu.Items.FindByText("Save").Enabled = true;
            ASPxButton C3 = (ASPxButton)SP.FindControl("btnSave");
            if (C3 != null)
            {
                ((IPostBackEventHandler)C3).RaisePostBackEvent(null);
            }
        }
        if (e.Item.Text == "Load")
        {
```

```
            HeaderMenu.Items.FindByText("Run").Enabled =
HeaderMenu.Items.FindByText("Clear").Enabled =
HeaderMenu.Items.FindByText("Save").Enabled = true;
            ASPxButton C4 = (ASPxButton)SP.FindControl("btnLoad");
            if (C4 != null)
            {
                ((IPostBackEventHandler)C4).RaisePostBackEvent(null);
            }
        }
        if (e.Item.Text == "Run")
        {
            HeaderMenu.Items.FindByText("Run").Enabled =
HeaderMenu.Items.FindByText("Clear").Enabled =
HeaderMenu.Items.FindByText("Save").Enabled = true;
            ASPxButton C5 = (ASPxButton)SP.FindControl("btnRun");
            if (C5 != null)
            {
                ((IPostBackEventHandler)C5).RaisePostBackEvent(null);
            }
        }
    }
}
------------------------------------------------------------
--  JAVA SCRIPT CODE FOR MASTER PAGE
------------------------------------------------------------
var HeroDemo = {
        // Site master
ClientLayoutSplitter_Init: function(s, e) {
    document.getElementById("HeaderArea").style.height =
    s.GetPaneByName("HeaderPane").GetClientHeight() -
ClientHeaderMenu.GetMainElement().offsetHeight + "px";

s.GetPaneByName("CornerPane").SetSize(ClientCornerMenu.GetMainElement().offsetHeight);
},

ClientInfoMenu_ItemClick: function (s, e) {
    if (e.item.parent && e.item.parent.name == "theme") {
        ASPxClientUtils.SetCookie("MailDemoCurrentTheme", e.item.name || "");
        e.processOnServer = true;
    }
},
       // Search Box
ClientSearchBox_TextChanged: function (s, e) {
    MailDemo.OnSearchTextChanged();
},
OnSearchTextChanged: function () {
    window.clearTimeout(HeroDemo.searchBoxTimer);
    var searchText = ClientSearchBox.GetText();
    if (ClientHiddenField.Get("SearchText") == searchText)
        return true;
    ClientHiddenField.Set("SearchText", searchText);
},
ClearSearchBox: function () {
    ClientHiddenField.Set("SearchText", "");
    ClientSearchBox.SetText("");
},
};
```