UNIVERSITE SAAD DAHLAB DE BLIDA

Faculté des Sciences

Département d'Informatique

MEMOIRE DE MAGISTER

Spécialité : Systèmes d'information et des connaissances

Une ontologie pour l'aide à la conception automatisée d'un entrepôt de données XML

Par

M^{r} HAMADI Abdelkrim

Devant le jury composé de :

Benblidia Nadjia	M.C A, U.de Blida	Présidente
Zeggour D.Eddine	Professeur, Ecole Supérieure d'informatique	Examinateur
Ahmed Nacer	Professeur, USTHB	Examinateur
Oukid Khouas Saliha	M.C A, U.de Blida	Promotrice
Omar Boussaid	Professeur, U.Lyon 2 France	Co-Promoteur

Blida, juillet 2011

RESUME

Une grande quantité de données stockée sous le format XML *(Extensible Markup Language)* est nécessaire pour l'aide à la décision. On rencontre ce format de données dans le e-commerce et les échanges d'informations sur Internet. De ce fait l'intégration de XML dans les entrepôts de données devient nécessaire. On trouve dans la littérature plusieurs approches pour automatiser la conception d'un entrepôt de données multidimensionnel cependant les règles d'automatisation ne sont pas bien identifier, définis et formaliser.

Dans notre travail nous nous intéressons aux entrepôts de données XML (données complexes), nous proposons une représentation des connaissances *(Knowledge Representation (KR))* pour l'aide à la conception automatisée d'un entrepôt de données XML. Cette représentation des connaissances formalisée en ontologie permettra de partager et de réutiliser les connaissances pour automatiser la modélisation d'un entrepôt de données XML de manière non ambiguë. Dans une première étape, on définira l'ontologie « XML-DFED_ONTO » qui regroupera les concepts existants dans un schéma XML, une dépendance fonctionnelle ainsi que les concepts d'un entrepôt de données. Cette ontologie décrira les vocabulaires, les terminologies et les relations entre ces concepts. L'étape suivante consistera à coder l'ontologie en utilisant la technologie du web sémantique qui est le OWL *(Ontology Web Language)*, un langage pour coder les ontologies sur le web. Un enrichissement de cette formalisation sera fait en définissant des contraintes sur les concepts de l'ontologie en utilisant le SWRL *(Semantic Web Rule Language)*, un langage de règles basées sur OWL. L'interrogation de la base des connaissances se fera en utilisant le langage SQWRL *(Semantic Query-Enhanced Web Rule Language)*.

En final, on implémentera cette ontologie en utilisant le logiciel Protégé et on vérifiera sa consistance avec le moteur d'inférence Pellet, ensuite l'ensemble des concepts de l'ontologie codée en OWL ainsi que les contraintes SWRL seront mappées en faits JESS et en règles JESS respectivement, ce qui nous permettra d'utiliser le moteur d'inférence JESS *(Java Expert System Shell)* pour faire du raisonnement sur la base des connaissances.

Mots clés : Entrepôt de donnée, représentation des connaissances, XML, schéma XML, dépendance fonctionnelle, ontologie, OWL, SWRL, SQWRL, Pellet, JESS.

الملخص

كمية كبيرة من البيانات المخزنة تحت شكل XML ضرورية في ميدان Business intelligence، نجد هذا الشكل من البيانات في التجارة الإلكترونية وتبادل المعلومات على شبكة الانترنت, إذن إدماج XML في مستودعات البيانات أصبح ضرورة.نجد في البحوث عدة طرق لجعل تصميم مستودع البيانات متعدد الأبعاد أوتوماتيكيا ولكن قواعد التصميم أللأوتوماتيكي ليست واضحة و غير معينة بصفة رسمية.

في هذه المذكرة، نوجه اهتمامنا إلى مستودع البيانات XML متعدد الأبعاد، نقترح تمثيل للمعرفة الخاصة بالتصميم أللأوتوماتيكي لهذا المستودع. تكون هذه المعرفة ممثلة على شكل أونتولوجي، ستمكننا من تقاسم استعمال تمثيلية المعرفة بطريقة واضحة.في مرحلة أولى، سنقوم بتصميم أونتولوجي XML-EDDF_ONTO التي ستحوي المفاهيم الموجودة في XML Schema، functional dependence و مستودع البيانات. هذه أللأونتولوجي ستوصف المفردات ، المصطلحات و العلاقات الموجودة مابين هذه المفاهيم. المرحلة التالية تتمثل في ترميز هذه أللأونتولوجي بلغة OWL و SWRL.سنقوم بالبحث و استنتاج مفاهيم جديدة باستعمال لغة SQWRL.في الأخير، سنستعمل البرنامج Protege لتنفيذ هذه أللأونتولوجي و محرك الاستدلال Pellet لتحقق من التركيبة الجيدة للأونتولوجي. كل مفاهيم أللأونتولوجي الممثلة بلغة OWL و SWRL ستحول إلى عناصر JESS، هذا التحويل سيسمح لنا من الوصول إلى استنتاجات معرفية جديدة.

الكلمات الرئيسية : مستودع البيانات، تمثيلية المعرفة ، XML، XML Schema، functional dependence، أونتولوجي ، OWL، SWRL، SQWRL، Pellet، JESS.

ABSTRACT

A big quantity of data stored under the XML format is necessary for the business intelligence, we meet this format of data in the e-commerce and the information exchanges on Internet, so the integration of XML in the datawarehouse becomes necessary.
We find in the literature several approaches to automate the conception of a multidimensional datawarehouse however the rules of automation are not clearly identify, define and formalize.

In our work, we are interested to the XML datawarehouse, we propose a knowledge representation (KR) to assist an automatic design of XML datawarehouse. This knowledge representation will allow to share and reuse the knowledge in a not ambiguous way. In the first step, we define the ontology “XML-DFED_ONTO” that will regroup the all concepts of XML Schema, the functional dependence and datawarehouse concepts. This ontology will describe vocabularies, terminologies and relations between these concepts. The following step consists to code this ontology with the OWL and we will define the set of the constraints with the SWRL. We will construct some requests with the SQWRL.In finally, we will implement this ontology with Protege software and we will check the consistence with Pellet *(OWL reasoner)*, we will transform all OWL concepts and SWRL constraints under JESS facts and JESS constraints respectively so we will use JESS reasoner.

Key words: Datawarehouse, knowledge representation, XML, XML Schema, functional dependence, ontology, OWL, SWRL, SQWRL, Pellet, JESS.

DEDICACES

A mes parents dont le rêve était toujours de me voir réussir. Qu'ils sachent que leur place est dans mon cœur et ma pensée,
A mes chers frères Mehdi et Riad pour leur soutien dans la vie.

A mes grands parents, mes tantes, mes oncles et leurs familles, que ces modestes lignes leur servent de témoignage de mon attachement indéfectible au lien sacré de la famille.

A mes neveux et nièces.

A tous mes enseignants depuis le primaire jusqu'au magistère pour le savoir et les connaissances qu'ils m'ont inculqué.

A tous mes amis.

A tous ceux qui me sont chers.

REMERCIEMENTS

Au terme de ce travail, il m'est particulièrement agréable d'exprimer ma reconnaissance et mes vifs remerciements à toutes les personnes qui ont contribué, de près ou de loin, à sa réalisation. Principalement :

Dr. Oukid Khouas Saliha et le Pr. Omar Boussaid d'avoir bien voulu encadrer ce travail et mettre à ma disposition toutes les ressources nécessaires pour la réalisation de ce mémoire. J'ai toujours pu compter sur leur indéfectible appui ; j'ai trouvé auprès d'eux une écoute sans faille. Leur expérience, leurs qualités humaines m'ont été d'une grande utilité.

Aux membres de jury d'avoir bien voulu évaluer ce mémoire.

Mes sincères remerciements à tous les enseignants ayant contribué de prêt ou de loin à ce travail.

Mes plus profonds remerciements pour mes amis de l'école doctorale.

Enfin, je formule mes remerciements à toutes les personnes qui m'ont aidé à la réalisation de ce travail et qui sont si nombreuse pour en faire une liste nominative.

Merci à toutes et à tous.

« Des chercheurs qui cherchent, on en trouve.
Des chercheurs qui trouvent, on en cherche »

Charles de Gaulle

TABLE DES MATIERES

LISTE DES FIGURES

LISTE DES TABLEAUX

LISTE DES EQUATIONS

INTRODUCTION

1. Contexte

Dans un contexte organisationnel de plus en plus complexe et mouvant, les entreprises sont souvent confrontées à une concurrence forte et à des clients de plus en plus exigeants. Pour faire face aux enjeux économiques, les entreprises doivent disposer d'informations et d'indicateurs pertinents permettant une vision stratégique claire et une anticipation des décisions majeures, indispensables à leur survie. D'un autre côté, le capital d'information dont disposent les entreprises est de plus en plus important et varié. Bien que les systèmes d'information (SI) classiques, considérés comme levier de la performance et de la compétitivité, permettent la gestion de cette masse importante de données. Toutefois, ils ne sont généralement pas organisés dans une perspective décisionnelle. Les données gérées par ces SI sont souvent éparpillées sur plusieurs bases hétérogènes et ne permettent pas d'avoir une vue transversale de l'activité de l'entreprise. De ce fait, il devient fondamental de les rendre homogènes et de les rassembler afin qu'elles facilitent la prise de décision. C'est là, l'objectif des entrepôts de données, qui se trouvent au cœur de l'architecture d'un système d'information décisionnel.
Un entrepôt de données (ED) se définit comme "une collection de données intégrées, orientées sujet, non volatiles, historisées, résumées et disponibles pour l'interrogation et l'analyse" **[3]**. Il constitue une extension aux bases de données (BD) classiques. En fait, les domaines de ces deux technologies ont en commun le traitement de grands volumes de données. Cependant, leurs missions et orientations sont complètement différentes. En effet, les BD sont destinées à fournir des outils de traitement automatisé pour répondre à des interrogations relativement standardisées (courtes durées, petit nombre d'enregistrements manipulés etc.) tandis que les ED sont destinés à stocker toutes les données utiles à la prise de décision.

Les problèmes posés par la construction d'un ED touchent essentiellement la définition de son schéma ainsi que son chargement. En raison des divergences, déjà citées, entre les ED et les BD, les approches de conception adoptées pour ces dernières s'avèrent inadaptées pour les ED **[94]**. En effet, la conception des BD se base totalement sur la phase de spécification des besoins (exprimés par un cahier de charges) pour définir le modèle conceptuel. Tandis que, la conception des ED nécessite les besoins des utilisateurs, la structure des sources de données d'alimentation, ou de ces deux composantes simultanément. Par conséquent, le besoin d'une méthode de modélisation spécifique pour les entrepôts de données s'avère indispensable. Ceci a motivé l'émergence des approches de conception appropriées pour les ED durant la dernière décennie.

La conception du schéma d'ED est une tâche complexe. Elle exige plusieurs efforts et une bonne expertise du concepteur. Cette expertise doit obligatoirement être affirmée, tant au niveau des modèles des sources de données qu'en compétence en définition des besoins analytiques.
Dans la littérature, les approches de conception des entrepôts de données sont classées en trois catégories : les approches dirigées par les sources de données (ascendantes), les approches dirigées par les besoins des utilisateurs (descendantes) et les approches mixtes.
Les approches dirigées par les sources **[18] [19] [31] [32]** privilégient la structure des sources dans leur processus de conception. Les approches de cette catégorie conçoivent le schéma de l'ED en partant d'une analyse détaillée des modèles de données des sources opérationnelles (généralement un diagramme E/A, une DTD ou un schéma XML) et ignorent les besoins d'analyse.
Quant aux approches dirigées par les besoins **[7] [33]**, elles suivent le principe de conception des BD classiques en partant des besoins des futurs utilisateurs du système décisionnel. Une fois collectés, ces besoins permettent d'élaborer un schéma conceptuel théorique (i.e., indépendant de toutes sources). Ces deux types d'approches sont complémentaires et peuvent être employées conjointement pour un meilleur résultat conceptuel, ce qui a donné naissance à une troisième catégorie d'approches dites mixtes **[34] [35] [36] [37]**. Cette dernière catégorie d'approches est celle qui fait l'objet de plus d'investigations actuelles au sein de la communauté scientifique.

C'est dans ce contexte que s'articulent les travaux du mémoire que nous présentons. Il s'agit précisément de proposer une représentation des connaissances pour l'aide à la conception automatisée d'entrepôt de données complexes, on s'intéresse aux données XML vue qu'une grande quantité de données stockée sous le format XML est nécessaire pour l'aide à la décision, on rencontre ce format de données dans le e-commerce et les échanges d'informations sur Internet, de ce fait l'intégration de XML dans les entrepôts de données devient nécessaire. Ainsi, l'enjeu réside dans le fait de capturer toutes les règles qui sont utilisées dans les approches d'aide à une conception automatisée d'un entrepôt de données puis de les formaliser en une ontologie de connaissances. Nous nous intéresserons beaucoup aux approches de conception des ED dirigées par les sources de données, vu que ces dernières sont automatisables.

2. Problématique et objectifs

On trouve dans la littérature plusieurs approches qui ont pour but l'automatisation de la conception d'un entrepôt de données multidimensionnel. Cependant, les règles d'automatisation ne sont pas bien identifiées, ni bien définies et ni bien formalisées. C'est dans ce contexte que se positionnent les travaux que nous présentons dans ce mémoire. Il s'agit précisément de chercher toutes ces connaissances de conception pour définir une représentation des connaissances, puis formaliser cette dernière en une ontologie. Pour concevoir cette dernière, nous allons dans un premier temps extraire toutes les connaissances relatives aux approches d'aide à la conception automatisée d'un entrepôt de données **[95] [96] [97] [98]**. Puis nous organiserons ces

connaissances dans une structure hiérarchique tout en essayant d'intégrer des ontologies déjà existantes, comme l'ontologie globale OLAP **[100]**. Dans notre implémentation, nous utiliserons le langage OWL **[102] [104]** qui est un langage de représentation d'ontologie sur le web, recommandé par le World Wide Web Consortium (W3C). Comme ce langage est basé sur les logiques descriptives **[101]**, il ne peut pas y avoir d'ambiguïté dus à l'inconsistance de la terminologie. Il est employé dans notre étude pour décrire les connaissances qui concernent le domaine de l'automatisation d'une conception d'un entrepôt de données XML.

Malgré que le langage OWL soit bien adapté pour représenter la structure des connaissances telle que les classes, les propriétés et les taxonomies, il ne peut pas représenter une connaissance déductive, qui est sous forme d'une règle. De ce fait, le SWRL **[103]**, qui est un langage de règles basé sur OWL, sera utilisé pour exprimer divers contraintes sur les connaissances.
Nous coderons cette ontologie ainsi que les règles SWRL en utilisant l'éditeur Protégé **[88]**, et nous utiliserons Pellet **[92]** comme outil de vérification et SQWRL comme moteur de requêtes. Nous utiliserons également le système JESS **[105]**, comme moteur de règles en transformant l'ontologie exprimée en OWL ainsi que les contraintes SWRL en faits JESS et règles JESS respectivement.

3. Structure du mémoire

Ce mémoire est organisé en six chapitres :

- Le chapitre 1 présente un panorama des principaux concepts liés aux entrepôts de données.
- Dans le chapitre 2, nous présentons la technologie XML vue que notre étude portera sur les entrepôts à données XML.
- Le chapitre 3 présente une définition complète de l'ontologie, sa structuration, son cycle de développement, ses domaines d'intérêts, ses langages de codage et ses outils.
- Le chapitre 4 est consacré à un état de l'art sur les approches de conception d'entrepôts de données. Nous y détaillons les principaux travaux proposés dans la littérature et qui sont relatifs à notre problématique.
- Le chapitre 5 détaille l'approche proposée pour automatiser la conception d'un entrepôt de données XML, on présente la conception et l'implémentation de la représentation des connaissances, cette représentation sera formalisée en une ontologie.
- Le chapitre 6 présente l'éxécution d'un test de conception d'un entrepot de données à partir d'un éxemple de schéma XML en utilisant notre approche qui est basée sur une ontologie de connaissances XML-DFED_ONTO.

Nous dressons, à la fin de ce mémoire, un bilan de nos travaux et nous présentons les perspectives de recherche envisagées.
En outre, pour faciliter la lecture du présent rapport, les détails de la représentation des connaissances proposée sont présentés dans l'annexe.

CHAPITRE 1
LES ENTREPOTS DE DONNEES

A travers ce chapitre, nous allons présenter les différents concepts liés au domaine des entrepôts de données.

1.1. Introduction

En raison de la mondialisation, les entreprises doivent être en mesure de livrer une vive concurrence qui est de plus en plus complexe et changeant sur les marchés mondiaux, les entreprises sont confrontées à une concurrence de plus en plus forte, des clients de plus en plus exigeants, pour cela les entreprises doivent s'appuyer sur des informations pertinentes, pour être efficace et faire face aux nouveaux enjeux économiques.
Mais souvent, les données dans les entreprises sont éparpillées dans de multiples systèmes hétérogènes et ne sont pas organisées dans une vision décisionnelle. Il est alors indispensable de les homogénéiser et de les regrouper afin de faciliter la prise de décision, cela ne peut ce faire sans la définition d'une architecture qui serve de fondations aux applications décisionnelles, d'où le nouveau concept «l'informatique décisionnelle».

L'informatique décisionnelle est un secteur en plein développement. La demande de systèmes d'aide à la décision *(SAD)* et de systèmes interactifs d'aide à la décision *(SIAD)* est de plus en plus forte, au vu de la croissance exponentielle des données manipulées par les entreprises.

1.2. Transactionnel & décisionnel

1.2.1 Système d'information (SI)

Le système d'information transactionnel est composé de l'ensemble des composants et d'applications métiers de l'organisation qui gèrent les données au quotidien. Il assure la gestion de toutes les transactions qui ont lieu au sein de l'organisation. Ces systèmes ne sont pas adaptés pour faire des analyses complexes de données **[1]** et ne préparent pas les données pour la prise de décision. Pour assurer une plus grande réactivité et une plus grande compétitivité, Les approches traditionnelles s'avèrent donc rapidement insuffisantes et les décideurs requièrent des systèmes qui facilitent leur processus de prise de décision.

Définition d'un système d'information : «Le système d'information est l'ensemble des méthodes et moyens recueillant, contrôlant et distribuant les informations

nécessaires à l'exercice de l'activité en tout point de l'organisation. Sa fonction est de produire et de mémoriser les informations, représentation de l'activité du système opérant (système opérationnel), puis de les mettre à disposition du système de décision (système de pilotage)» **[2]**.

1.2.2 Système d'information décisionnel

Le système d'information décisionnel est un ensemble organisé d'information, facilement accessible, qui est adapté pour la prise de décision. Il est obtenu à partir de traitements sur les données de systèmes d'information transactionnels internes ou externes à l'organisation.

Définition d'un système d'information décisionnel : est un système qui réalise la collecte, la transformation des données brutes issues de sources de données et le stockage dans d'autres espaces ainsi que la caractérisation des données résumées en vue de faciliter le processus de prise de décision. Ce qui caractérise les systèmes d'information décisionnel, c'est la possibilité de poser une grande variété de questions au système, certaines prévisibles et planifiées comme des tableaux de bord et d'autres imprévisibles, et de permettre à l'utilisateur d'effectuer les requêtes qu'il souhaite, par lui-même, sans l'intervention de programmeurs. Ils doivent assurer une cohérence globale des données. Pour ce faire, leur alimentation doit être une opération réfléchie et planifiée dans le temps. Les transferts de données du système opérationnel vers le système décisionnel seront réguliers avec une périodicité bien choisie dépendante de l'activité de l'entreprise. A l'inverse des systèmes transactionnel, aucune information n'y est jamais modifiée.
Les deux tableaux suivants montrent la différence entre les systèmes transactionnels et ceux décisionnels.

Système transactionnel	Système décisionnel
Orienté applications	Orienté thèmes et sujets
Situation instantanée	Situation historique
Données détaillées et non redondantes	Informations agrégées cohérentes souvent avec redondance
Données changeant constamment	Informations stables et synchronisées dans le temps
Pas de référentiel commun	Un référentiel unique

Tableau 1.1 : Transactionnels Vs SID : La différence par les données.

Système transactionnel	Système décisionnel
Assure l'activité au quotidien	Permet analyse et prise de décision
Pour les opérationnels	Pour les décideurs
Mise à jour et requêtes simples	Lecture uniquement et requêtes complexes transparentes
Temps de réponse immédiat	Temps de réponse moins critique
Faible volume à chaque transaction	Large volume manipulé
Conçue pour la mise à jour	Conçue pour l'extraction
Usage maitrisé	Usage aléatoire

Tableau 1.2 : Transactionnels Vs SID : La différence par l'usage.

En résumé, le SID se construit à partir des données opérationnelles du système d'information classique qui sont souvent stockées dans des systèmes hétérogènes. Il récupère ces données provenant de ces sources et les prépare pour le système de décision. Pour ce faire, le système d'information décisionnel va remplir trois fonctions : extraction des données, leur stockage et la restitution des données sous une forme exploitable. Le stockage sert à structurer les données au sein d'un entrepôt de données. Il s'agit de mettre en place un schéma relationnel orienté décisionnel, ce dernier constitue la partie la plus important et la plus complexe du processus de développement des SID. C'est la raison pour laquelle, nous nous intéressons à cette partie du système d'information décisionnel.
De ce fait, il est nécessaire de construire une structure pour les héberger, les organiser et les restituer à des fins d'analyse. Cette structure est l'entrepôt de données ou «Datawarehouse».

1.3. Le Datawarehouse

Le concept d'entrepôt de données a pris forme au début des années 90, il est devenu depuis la clé de voûte de ce que l'on appelle l'informatique décisionnelle. La vogue grandissante de ce concept est le fruit de l'évolution de l'informatique dans les organisations.
Il s'agit en fait de mettre à la disposition des gestionnaires tout ou une partie des données qui ont été accumulées dans les fichiers et bases de données des systèmes transactionnels depuis de nombreuses années et qui ne peuvent être exploitées de manière efficace. L'acquisition de ces données a coûté cher aux entreprises et leurs gestionnaires sont de plus en plus conscients de ce qu'ils perdent en n'y ayant pas accès pour prendre leurs décisions. En effet, ces données recèlent des informations cruciales aussi bien sur le plan stratégique que sur le plan opérationnel. On parle alors de la transformation des données en informations.

1.3.1 Définition

De nombreuses définitions ont été proposées, soit académiques, soit par des éditeurs d'outils, de bases de données ou par des constructeurs, cherchant à orienter ces définitions dans un sens mettant en valeur leur produit.
Une définition des entrepôts de données a été proposée par Inmon et Hackarton en 1994 : «A data warehouse is a subject-oriented, integrated, time-variant, non-

volatile collection of data used in support of management decision making processes» **[3]**.
On peut traduire cette définition comme suit : «Les données d'un entrepôt de données sont intégrées, orientées sujet, non volatiles, historiées, résumées et disponibles pour l'interrogation et l'analyse» **[4]**.
L'objectif des entrepôts de données est d'offrir un accès aux données même si la source de ces donnés est inaccessible, limitant par là les accès distants aux systèmes gérant les données d'origines.
Le rôle principal de l'administrateur est de minimiser et de réduire ces accès aux sources de données qui ont la particularité d'être volumineuses et hétérogènes grâce à l'utilisation de techniques d'optimisation de requêtes.
Parmi ces techniques, nous pouvons citer les indexes et les vues matérialisées. Il existe d'autres techniques telles que la fragmentation, le clustering et le traitement parallèle des requêtes.

1.3.2 Caractéristiques des entrepôts de données

«Un entrepôt est une collection de données orientées sujet, intégrées, non volatiles et historiées, organisées pour le support d'un processus d'aide à la décision.» Bill Inmon.
Analysons point par point cette définition, les données d'un entrepôt sont donc :

- Orientées sujet : Les données sont orientées métier et donc triées par thèmes. L'intégration dans une structure unique est indispensable pour éviter aux données concernées par plusieurs sujets d'être dupliquées. Cependant, dans la pratique, il existe également des entrepôts plus petits, les magasins de données (data marts) ou l'entrepôt est fragmenté en plusieurs bases qui sont orientées sujet.
  L'intérêt de cette organisation est de pouvoir disposer de l'ensemble des informations utiles sur un sujet le plus souvent transversal aux structures fonctionnelles et organisationnelles de l'entreprise.

- Intégrées : Les données proviennent de plusieurs sources hétérogènes et disparates. Avant d'être intégrées dans l'entrepôt, les données doivent êtres mises en forme et unifiées afin d'avoir un état cohérent. Cela nécessite un gros travail de normalisation, de gestion des référentiels et de cohérence. Une donnée doit avoir une description et un codage unique. Cette phase d'intégration ou de nettoyage des données est très complexe et représente 60 à 90% de la charge totale d'un projet.

- Non volatiles : Les données sont stables et non modifiables. Un entrepôt de données doit garantir qu'une requête lancée à différentes dates sur les mêmes données donne toujours les mêmes résultats. De plus, les données d'un entrepôt sont mises à jour périodiquement, ce ne sont donc pas des informations en temps réel.

- Historiées : Les données sont historiées donc datées. L'historisation est nécessaire pour suivre dans le temps l'évolution des différentes valeurs des indicateurs à analyser. Ainsi, un référentiel temps doit être associé aux données afin de permettre l'identification de valeurs précises dans la durée.

- Aide à la décision : Un entrepôt de données est destiné à l'aide à la décision, ce qui fait que les traitements qui s'y appliquent sont de nature différente de ceux des systèmes transactionnels, on parle ainsi de OLAP par opposition à OLTP. La plupart des traitements transactionnels en ligne n'impliquent que quelques données, occasionnent des changements dans la base de données et requièrent une réponse presque instantanée alors que les traitements mis en œuvre pour l'aide à la décision impliquent la lecture de nombreuses données mais n'entraînent pas de changement dans la base de données et ne requièrent pas une réponse instantanée.

1.3.3 Architecture d'un entrepôt de données

L'architecture des entrepôts de données repose souvent sur un SGBD séparé du système de production de l'entreprise qui contient les données de l'entrepôt. Le processus d'extraction des données permet d'alimenter périodiquement ce SGBD. Néanmoins avant d'exécuter ce processus, une phase de transformation est appliquée aux données opérationnelles, celle-ci consiste à les préparer (mise en correspondance des formats de données), les nettoyer, les filtrer,..., pour finalement aboutir à leur stockage dans l'entrepôt.
La figure 1.1 qui suit illustre le schéma conceptuel d'un entrepôt de données, ce dernier stocke des données brutes ou modifiées issus de sources de données hétérogènes et distribuées.

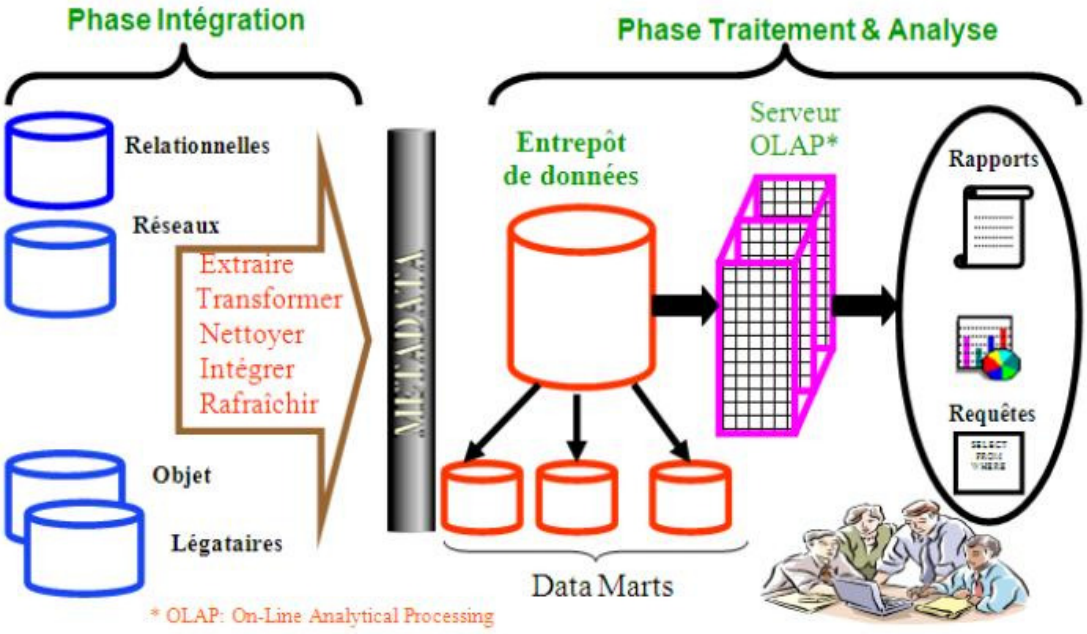


Figure 1.1 : Schéma conceptuel d'un entrepôt de données [7].

a) Les sources : Les données de l'entrepôt sont extraites de diverses sources souvent réparties et hétérogènes, qui doivent être transformées avant leur stockage dans l'entrepôt. Les données sources peuvent être interne a l'entreprise, ces données sont saisies à partir des différents systèmes de production qui rassemblent les divers SGBD opérationnels, ainsi que des anciens systèmes de production qui contiennent des données encore

exploitées par l'entreprise ou externe a l'entreprise souvent fournie par une tierce partie.

b) L'entrepôt de données: C'est le noyau, il contient un ensemble de « data-mart » représentant chacun une activité ou une fonctionnalité d'une organisation. Il existe plusieurs types de données dans un entrepôt qui correspondent à diverses utilisations :

- Données de détail courantes : Ce sont l'ensemble des données quotidiennes et plus couramment utilisées. Ces données sont généralement stockées sur le disque pour avoir un accès rapide.
- Données de détail anciennes : Ce sont des données quotidiennes concernant des événements passés, comme par exemple le détail des ventes des deux dernières années.
- Données résumées ou agrégées : Ce sont des données moins détaillées que les deux premières et elles permettent de réduire le volume des données à stocker.
- Les métadonnées : Ce sont des données essentielles pour parvenir à une exploitation efficace du contenu d'un entrepôt. Elles représentent des informations nécessaires à l'accès et l'exploitation des données dans l'entrepôt comme : la sémantique (leur signification), l'origine (leur provenance), les règles d'agrégation (leur périmètre), le stockage (leur format, par exemple : francs, euro,...) et finalement l'utilisation (par quels programmes sont-elles utilisées).

c) Outils : Il existe sur le marché différents outils pour l'aide à la décision, comme les outils de fouille de données ou datamining (pour découvrir des liens sémantiques), outils d'analyse en ligne (pour la synthèse et l'analyse des données multidimensionnelles), outils d'interrogation (pour faciliter l'accès aux données en fournissant une interface conviviale au langage de requêtes).

1.3.4 Entrepôt de données & Base de données

Dans l'environnement des entrepôts de données, les opérations, l'organisation des données, les critères de performance, la gestion des métadonnées, la gestion des transactions et le processus de requêtes sont très différents des systèmes de bases de données opérationnels. Par conséquent, les SGBD relationnels orientés vers l'environnement opérationnel, ne peuvent pas être directement transplantés dans un système d'entrepôt de données **[5]**.
Les SGBD ont été crées pour les applications de gestion de systèmes transactionnels. Par contre, les entrepôts de données ont été conçus pour l'aide à la prise de décision. Ils intègrent les informations qui ont pour objectif de fournir une vue globale de l'information aux analystes et aux décideurs.
Le tableau suivant résume ces différences entre les systèmes de gestion de bases de données et les entrepôts de données **[6]**.

Caractéristique	Base de données (Système opérationnel)	Entrepôt de données (Système informationnel)
Raison d'être	Supporter les opérations courantes de l'entreprise	Supporter le processus de décision des gestionnaires
Type de données	Représentation courante de l'état de l'entreprise	Historiques au moment précis dans le temps
Principaux utilisateurs	Commis, vendeurs, administrateurs et employés	Gestionnaires, analystes, clients et décideurs
Envergure d'utilisation	Etroit, mises à jour et requêtes simples	Large, analyses et requêtes complexes
But de la conception	Performance	Facilité d'accès et d'utilisation
Taille	Méga octets	Téra octets
Nombre d'utilisateurs	Elevé	Petit
Mode d'accès	Lecture/Ecriture	Lecture

Tableau 1.3 : Comparaison entre une base de données et un entrepôt de données.

1.4. Modélisation multidimensionnelle

Suivant cette définition, les données à analyser au niveau d'un magasin doivent refléter la vision d'une classe d'analystes **[7] [8]**. Cette vision correspond à une structuration des données selon plusieurs axes d'analyses (ou dimensions) pouvant représenter des notions variées telles que le temps, la localisation géographique, le code identifiant des produits...
La modélisation multidimensionnelle consiste à considérer un sujet analysé comme un point dans un espace à plusieurs dimensions. Les données sont organisées de manière à mettre en évidence le sujet analysé et les différentes perspectives de l'analyse. Avant de décrire les différents schémas, nous commençons par quelques concepts de base.

1.4.1 Concepts de bases

Conceptuellement, cette modélisation multidimensionnelle a donné naissance aux concepts de fait et de dimension **[7]**.

- Concept de fait

Le sujet analysé est représenté par le concept de fait.
Définition : Le fait modélise le sujet de l'analyse. Un fait est formé de mesures correspondant aux informations de l'activité analysée. Les mesures d'un fait sont numériques et généralement valorisées de manière continue **[7]**. Par exemple, dans le fait Ventes, nous pouvons avoir la mesure "Quantité de produits vendus par magasin".

- Concept de dimension

Le sujet analysé, c'est à dire le fait, est analysé suivant différentes perspectives. Ces perspectives correspondent à une catégorie utilisée pour caractériser les mesures d'activité analysées **[8]**; on parle de dimensions.
Définition : Une dimension modélise une perspective de l'analyse. Une dimension se compose de paramètres correspondant aux informations faisant varier les mesures de l'activité.

- Concept hiérarchie

La hiérarchie sert lors des analyses pour restreindre ou accroître les niveaux de détail de l'analyse.
Définition : Une hiérarchie organise les paramètres d'une dimension selon une relation "est-plus-fin" conformément à leur niveau de détail (Ville Département Région).

La conception et la construction d'un entrepôt de données représente un enjeu important pour la communauté des bases de données. Cette conception doit concerner les trois niveaux de la modélisation classique, conceptuelle et logique puis physique qui consiste à proposer des techniques d'optimisation adaptées aux données XML.

1.4.2 Niveau conceptuel

Contrairement à la modélisation logique, Il n'existe à ce jour aucun consensus sur la modélisation conceptuelle des entrepôts de données comme cela peut être le cas avec la méthode MERISE pour la conception des bases de données relationnelles. Différentes travaux ont été proposées dans la littérature, ils y a ceux basées sur le modèle E/R (entité- association) **[9]**, **[10]**, **[11]** et **[12]** représentent les concepts décisionnels via des extensions plus ou moins importantes des concepts d'entités et d'associations, par exemple dans **[13]**, les auteurs proposent un modèle multidimensionnel appelé StarER qui a pour concepts de base des faits, des entités, des relations et des attributs.

D'autres approches basées sur les modèles orientées objet comme UML *(Unified Modeling Language)* **[14]** et **[15]** par exemple dans le papier **[16]** les auteurs proposent un profil (ensemble de stéréotypes, de valeurs typées et de contraintes) pour spécialiser UML à la modélisation multidimensionnelle. Le fait est représenté par une classe avec le stéréotype «*Fact*», une dimension est représentée par une classe avec le stéréotype «*Dimension*». Un autre modèle multidimensionnel appelé YAM2 proposé par **[17]** basé sur la spécialisation du méta-modèle du diagramme de classes UML.

Il y a une troisième catégorie de modèles qui sont spécifiques tels que ceux présentées dans **[18]**, **[19]**, **[20]**, **[21]**, **[22]**, **[23]**. Par exemple dans le papier **[18]** les auteurs proposent un modèle appelé " *Dimensional Fact Model* " (DFM). Cette modélisation multidimensionnelle graphique distingue clairement les concepts tels que les faits, les dimensions, les mesures et les hiérarchies. Cependant, les modèles spécifiques ne sont pas connus des concepteurs contrairement à ceux basées sur le modèle E/R et UML, cela peut constituer un obstacle.
Le Tableau suivant résume la classification des travaux traitant le problème de conception des entrepôts de données :

Travaux sur les modèles conceptuels	Les différents modèles	Observation
(Sapia et al. 1999) (Tryfona et al. 1999) (Sànchez et al. 1999) (Franconi et Sattler 1999) (Franconi et Kimble 2004b) (Trujillo et Palomar 1998)	Le modèle entité-association (E/R)	Le modèle le plus utilisé dans la conception des systèmes d'information. Les caractéristiques de la modélisation multidimensionnelle (Agrégation et spécialisation) n'existent pas.
(Prat and Akoka 2002) (Trujillo et al. 2002) (Lujàn-Mora et al. 2006) (Abello et al. 2002, 2006)	Le modèle orienté objets	Bénéficie de la puissance d'expression de l'UML.
(Golfarelli et al.1998b) (Cabibbo and Torlone 1998) (Moody and Kortink 2000) (Husemann et al. 2000) (Tsois et al. 2001) (Vassiliadis et al.2002a, 2005)	Le modèle spécifique	Utilise exclusivement des concepts multidimensionnels. Modèle non connu par les concepteurs, ce qui nécessite la maitrise de ces modèles.

Tableau 1.4 : les différents modèles utilisés dans la conception des entrepôts de données.

1.4.3 Niveau logique

Pour construire un modèle approprié pour un entrepôt de données, nous pouvons choisir, soit un schéma relationnel (le schéma en étoile, en flocon de neige ou en constellation) ou les données sont stockées dans un SGBD relationnel, soit un schéma multidimensionnel (Cube) ou les données sont stockées dans une base de données multidimensionnelle, ces différents schémas relèvent d'un niveau logique de conception car on a recours a la notion de table (table de faits, table de dimension).

1.4.3.1 Modèles en étoile (Star schema)

Ce modèle représente visuellement une étoile, on parle de modèle en étoile (*Star schema* **[7]**), ou tous les faits sont définis dans une simple table relationnelle.

Cette table va être reliée par sa clé primaire à d'autres tables correspondant aux dimensions.
Nous illustrons ce modèle dans la figure 1.2, le modèle en étoile essaie de superposer une structure multidimensionnelle au-dessus d'un modèle relationnel normalisé à deux dimensions. Le modèle en étoile simplifie le modèle logique normalisé en organisant les données de manière optimale pour les traitements d'analyse.

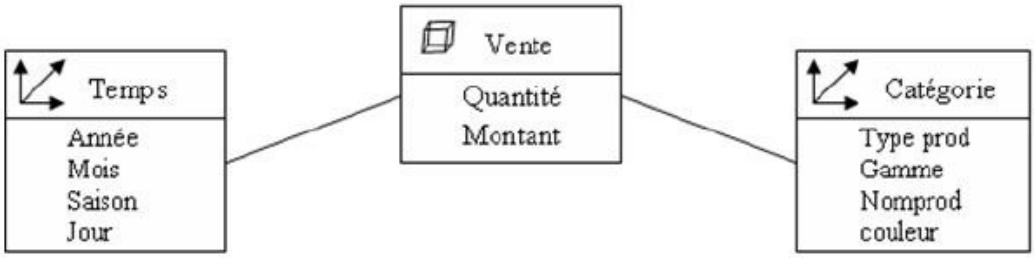


Figure 1.2 : Exemple d'un schéma en étoile [7].

1.4.3.2 Modélisation en flocon (Snowflake)

La modélisation en flocon est une modélisation en étoile pour laquelle nous éclatons les tables de dimension en sous-tables selon la hiérarchie de cette dimension comme illustré dans la figure 1.3.

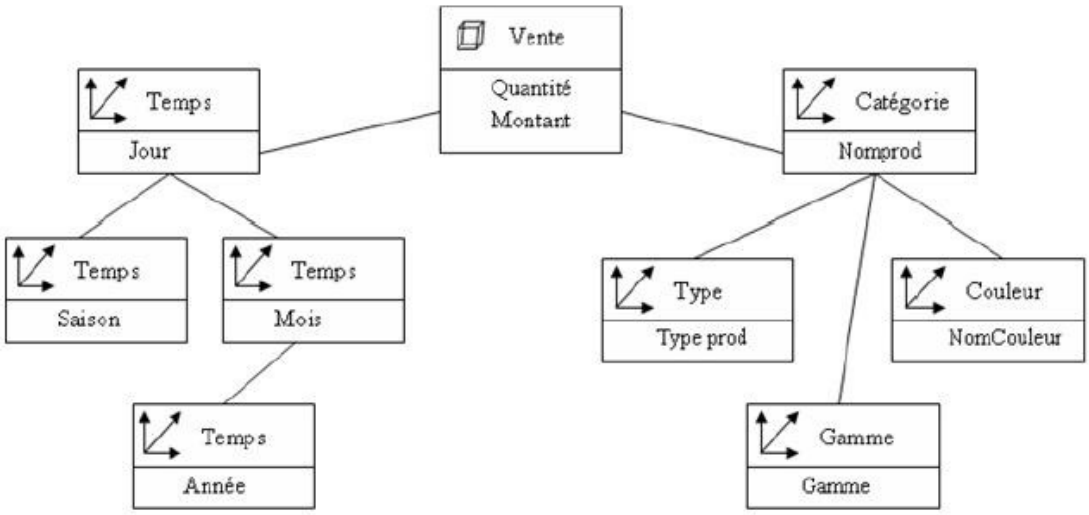


Figure 1.3 : Exemple d'un schéma en flocon de neige [7].

1.4.3.3 Modélisation en constellation

La modélisation en constellation. Il s'agit de fusionner plusieurs modèles en étoile qui utilisent des dimensions communes. Un modèle en constellation comprend donc plusieurs faits et des dimensions communes ou non.

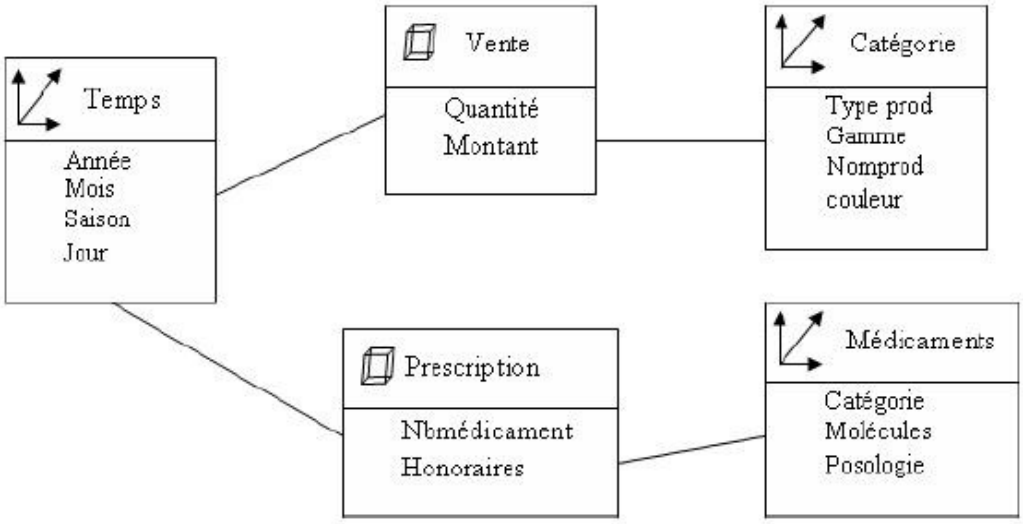


Figure 1.4 : Exemple d'un schéma en constellation [7].

1.4.3.4 Hyper cube

Les outils OLAP reposent sur des bases de données multidimensionnelles destinées à exploiter rapidement les dimensions d'une population de données. La plupart des solutions OLAP reposent sur un même principe : restructurer et stocker dans un format multidimensionnel les données issues de fichiers plats ou de bases relationnelles **[26]**. Ce format multidimensionnel connu sous le nom d'hyper cube permet aux utilisateurs d'analyser les données suivant des axes propres à leur métier.
Un hyper cube est un tableau à n dimensions. Chaque dimension possède une hiérarchie associée de niveaux de consolidation. Chaque position dans un tableau multidimensionnel correspondant à une intersection de toutes les dimensions, elle est appelée une cellule.
Ces dimensions peuvent être affinées, décomposées en hiérarchies, afin de permettre à l'utilisateur d'examiner ses indicateurs à différents niveaux de détail, de descendre dans les données, allant du niveau global au niveau le plus fin.

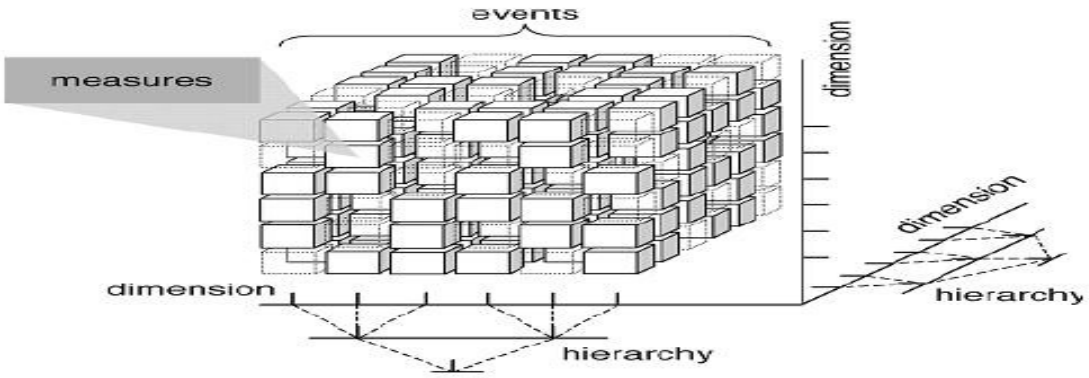


Figure 1.5 : Hyper cube [7].

1.4.4 Niveau physique

Le stockage de données et les méthodes d'accès associées (Les techniques d'indexation) ont été étudiés depuis plusieurs années. Cependant, les grands volumes gérés par les entrepôts et les besoins d'accès performant lors de l'interrogation ont remis en cause les techniques traditionnelles. Dans cette section nous présentons quelques techniques de stockage et d'accès aux données adaptées aux entrepôts.

1.4.4.1 Le stockage

Le stockage des modèles multidimensionnels (Les cubes) est réduit au problème de stockage d'un vecteur multidimensionnel, ou chaque dimension de cube est associée à un axe du vecteur et chaque valeur d'une dimension est associée à une position de l'axe correspondant à la dimension. Les cellules du vecteur, c'est-à-dire les intersections entre les positions des axes, contiennent les mesures.
Le problème principal posé par la représentation d'un cube sous la forme d'un vecteur multidimensionnel est que celui-ci est creux. Ceci est du au fait qu'en général seulement un nombre réduit de cellule d'un cube ont une valeur de mesure associées.

1.4.4.2 L'indexation

Les techniques d'indexation utilisées dans les bases de données de type (OLTP) ne sont pas bien adaptées aux environnements des entrepôts des données. En effet la plupart des transactions OLTP accèdent à un petit nombre de n-uplets, et les techniques utilisées (index B+ par exemple) sont adaptées à ce type de situation. Les requêtes décisionnelles adressées à un entrepôt de données accèdent au contraire à un très grand nombre de n-uplets (Ce type de requête est encore appelé requêtes d'intervalle).
Les différents types d'index utilisé dans les entrepôts de données : index binaire, index de jointure et index de jointure en étoile (star join index).

a) Index binaire : Un index binaire **[27]** est une séquence de bits où chacun d'eux indique si l'enregistrement associées présente une propriété donnée. Actuellement, presque tous les SGBDR commerciaux fournissent des indexes binaire. L'avantage des indexes binaire est qu'il est possible d'exécuter des opérations logiques (par exemple les opérations ET, OU, XOR, NOT) de manière performante **[27]**. L'autre avantage est que pour des attributs à nombre limités de valeurs, l'espace nécessaire pour leur stockage est réduit. Dans ce cas, l'index binaire peut être géré en mémoire, ce qui améliore les performances du système général. Des technique de compression de données sont utilisées pour gérer les indexes binaires qui contiennent presque dans leurs totalité des bits a zéro.

b) Index de jointure : Ce type d'index a été proposé par Valduriez appelé index de jointure (join indices) pour pré-joindre deux tables **[28]**. L'index matérialise les liens existants entre deux tables par le biais d'une table à

deux attributs contenant les identifiants des n-uplets joints. Cet index peut être vu comme une jointure pré calculée.

c) Index de jointure en étoile : Red Brick a proposé un index appelé index de jointure en étoile, adapté aux requêtes définies sur les entrepôts de données modélisés en étoile **[29]**. Un index de jointure en étoile peut contenir toute combinaison des identifiants des n-uplets de la table de faits et des identifiants des n-uplets des tables dimensions pouvant être jointes. Ce type d'index est dit complet s'il est construit en joignant toutes les tables de dimensions avec la table des faits. L'index de jointure en étoile complet est bénéfique à n'importe quelle requête définie sur un entrepôt modélisé en étoile.

1.4.4.3 Le partitionnement

Les entrepôts de données se prêtent bien à l'utilisation des techniques de partitionnement, car des requêtes complexes sur des grandes quantités de données peuvent être exécutées en parallèle améliorant éventuellement les temps de réponse.
Dans le cas des systèmes relationnels, les techniques de partitionnement de relations ont été étudiées depuis de nombreuses années. Les travaux récents dans le contexte des entrepôts de données visent au partitionnement vertical de la relation de fait d'un schéma en étoile. Dans le système multidimensionnel, il est possible de distinguer : La fragmentation horizontale se décline en deux versions : les fragmentations primaire et dérivée. La fragmentation primaire d'une relation est effectuée grâce à des prédicats de sélection définis sur la relation. La fragmentation horizontale dérivée s'effectue avec des prédicats de sélection définis sur une autre relation. La fragmentation horizontale apparaît plus spécialement appropriée. Une méthodologie de fragmentation horizontale pour décomposer un schéma d'entrepôt en étoile **[30]**, cette approche permet de fragmenter quelques/toutes les tables de dimension avec la fragmentation horizontale primaire, puis fragmenter la table des faits en utilisant les schémas de fragmentation des tables de dimension.

1.5. Approche de conception des entrepôts de données

Actuellement, plusieurs problèmes essentiellement de conception de ces entrepôts persistent. Ils sont dus à l'immaturité et à la complexité des méthodes et des outils de conception disponibles. Nous distinguons dans la littérature trois grandes approches de conception du schéma de l'entrepôt :

1.5.1 Basées sur les données sources

Schémas des bases de production qualifiée également d'ascendante. Ces méthodes partent du modèle du système d'information (SI) opérationnel de l'entreprise et appliquent un ensemble de règles qui permettent une succession des transformations sur le schéma source pour construire des schémas multidimensionnels **[18]**, **[19]**, **[31]** et **[32]**.

Par exemple, dans l'article **[18]** les auteurs proposent une méthodologie semi-automatique pour construire un schéma d'entrepôt de données à partir de schémas entité-relation qui représentent les bases de données sources. Le fait est défini comme une entité ou une association entre plusieurs entités fréquemment mises à jour et représentant un intérêt premier de l'organisation. Chaque fait devient la racine d'un arbre dont les nœuds sont les attributs de l'entité. Sur cet arbre, des élagages et des greffes des branches non guidés sont réalisés afin que les données contenues dans l'arbre soient pertinentes par rapport à l'application.

Puis, les dimensions sont définies en considérant les nœuds fils de la racine de l'arbre raffiné. Par la suite, les mesures des faits sont définies à partir des attributs numériques de l'arbre. Les inconvénients de ces méthodes sont :

- Les besoins OLAP *("On Line Analytical Processing ")* sont ignorés.
- L'évolution du schéma des sources de données.
- L'hétérogénéité et la complexité des sources de données.

1.5.2 Basées sur les besoins d'analyse

Approche orientée utilisateurs dénommée également descendante, elle définit le schéma de l'entrepôt en fonction des besoins d'analyse **[7]**, **[33]**. Les utilisateurs sont interrogés afin de collecter l'ensemble de leurs besoins d'analyse avant de construire l'entrepôt de données. Par exemple dans l'article de Tsois **[22]**, les auteurs ont proposé une méthode pour construire un schéma conceptuel basé sur les besoins d'analyse qui sont exprimés sous forme de requêtes. Le résultat des requêtes des utilisateurs est présenté sous forme tabulaire où les colonnes et les lignes sont composées des paramètres des dimensions. A partir de ces tableaux, les auteurs définissent un ensemble minimal de concepts (Niveau, hiérarchie, dimensions, cubes, attributs et des relations de manipulation de données).

Les limites de ces approches sont :

- La négligence du système opérationnel dit OLTP *(" On Line Transaction Processing ").*
- Dépend des besoins des personnes impliquées dans le processus de développement de l'entrepôt.
- Difficile de déterminer de façon exhaustive les besoins d'analyse pour l'ensemble des utilisateurs ainsi que leurs besoins futurs.

1.5.3 L'approche mixte

Elle est dite hybride, elle combine les deux approches précédentes, elle prend en compte à la fois les sources de données et les besoins d'analyse, en d'autre terme, elle construit des schémas multidimensionnels à partir des structures du système OLTP et les valident par rapport aux besoins analytiques **[34]**, **[35]**, **[37]** et **[36]**.

Par exemple dans le papier de Bonifati **[34]**, la méthode proposée définit le schéma de l'entrepôt suivant le modèle en étoile. Premièrement Elle commence à

générer manuellement un schéma idéal à partir des besoins utilisateurs, utilisant le paradigme GQM (Goal/Question/Metric), puis générer d'une manière automatique plusieurs schémas a partir du schéma de données et puis ces schémas sont confrontés à un seul schéma idéal (besoins).
Cette approche fait actuellement l'objet de beaucoup d'intérêt par la communauté des bases de données qui voient dans cette approche un consensus possible pour remédier aux insuffisances des deux méthodes citées précédemment. Cependant cette méthode a ses limites, elle hérite quelques inconvénients des méthodes précédentes tel que :

- La difficulté d'évaluer les besoins des utilisateurs et les schémas des sources de données.
- La complexité croissante des processus de conception dus au nombre important de schéma a généré a partir des sources et leurs comparaisons avec les besoins exprimés.

Le Tableau suivant présente une classification des travaux traitant le problème de la conception des entrepôts de données selon le type de l'approche (Basée sur les données sources, sur les besoins d'analyse ou sur l'approche hybride) :

Travaux	Approche de conception
(Cabibbo et al.,1998,2000) (Golfarelli et al.,1998a) (Hüsemann et al., 2000) (Golfarelli et al., 1998b) (Lechtenbörger et al., 2000) (Moody et al. 2000)	Orientée Données
(Kimball 96 et al. 98) (V Poe 1996) (Prakash et Gosain,2003 ; Schiefer, List et Bruckner, 2002) (Tsois et al.2001),	Orientées besoins d'analyse
(Bonifati, et al., 2001) (Phipps, et al., 2000) (Giorgini et al., 2005) (Soussi et al 2005) (Ghozzi et al. 2005)	Approche mixte

Tableau 1.5 : Classification des travaux selon le type de l'approche.

1.6. Implémentation des modèles multidimensionnels

Il existe actuellement trois approches pour l'implémentation d'un entrepôt de données: relationnelle OLAP (ROLAP), multidimensionnelle OLAP (MOLAP) et hybride OLAP (HOLAP).

- Dans l'approche ROLAP, les données sont organisées selon des schémas relationnels spécialisés (Figure 1.2, 1.3 et 1.4) (Flocon de neige, constellation, étoile.). Le langage de requêtes SQL et d'autres fonctionnalités des systèmes relationnels ont été adaptés à cette organisation.

- Dans l'approche MOLAP, les données sont organisées sous forme de cube. Les systèmes de bases de données qui supportent cette organisation offre un langage propriétaire pour l'interrogation et la manipulation de ces cubes.
- L'approche HOLAP, les données sont maintenues par un SGBD relationnel alors que les agrégations sont mises dans un SGBD multidimensionnel.

1.7. Grands projets de recherche

Dans cette partie, nous décrivons les projets de recherche sur les entrepôts de données.

1.7.1 DWQ

DWQ *(Foundations of Data Warehouse Quality)* **[38] [39]** est un projet de la communauté Européenne visant à développer des fondements sémantiques qui permettront d'aider les concepteurs d'entrepôts de données dans le choix des modèles, des structures de données avancées et des techniques d'implantation efficaces en s'appuyant sur des facteurs de qualité de service.

1.7.2 SIRIUS

Le projet SIRIUS *(Supporting the Incremental Refreshment of Information warehoUseS)* **[40]** développé à l'Université de Zurich, est un système d'entrepôt de données qui a pour objectif d'étudier des techniques permettant le rafraîchissement incrémental de l'entrepôt en réduisant les temps de mise à jour.

1.7.3 Squirrel

Squirrel **[41]**, **[42]**, **[43]** et **[44]** est un système de l'université du Colorado qui propose un cadre pour l'intégration de données basé sur la notion de médiateur d'intégration. Les médiateurs d'intégration sont des modules actifs qui supportent des vues intégrées à travers de multiples bases de données.

1.7.4 TSIMMIS

TSIMMIS *(The Stanford-IBM Manager of Multiple Information Sources)* **[45] [46] [46]** est un projet visant à fournir des outils pour un accès intégré à des entrepôts d'information. Chaque source d'information est équipée d'un traducteur qui encapsule la source, convertissant les objets sous-jacents dans un modèle de données commun.

1.7.5 WHIPS

WHIPS *(WareHouse Information Prototype at Stanford)* **[48] [49] [50]** est un système de gestion d'entrepôts de données utilisé comme banc d'essai. Le but de ce projet est de développer des algorithmes pour collecter, intégrer et maintenir

des informations émanant de sources hétérogènes, distribuées et autonomes. L'architecture du prototype WHIPS consiste en un ensemble de modules indépendants implantés comme des objets CORBA.

1.8. Conclusion

Les entrepôts de données et leurs outils deviennent indispensables dans le monde des entreprises. De nombreuses sources d'informations et de données semi structuré particulièrement XML sont échangées et existes dans les systèmes de production. Dans le chapitre suivant, nous allons introduire le langage XML. Tandis qu'une présentation de l'entrepot de données XML se fera au chapitre 4.

CHAPITRE 2
LA TECHNOLOGIE XML

Dans ce chapitre nous introduisons les principes du langage XML car les données en entrée de l'entrepôt de données complexes seront dans le format XML.

2.1. Le langage XML

XML est l'acronyme d'eXtensible Markup Language, il est un sous-ensemble du langage SGML1 *(Standard Generalized Markup Language)* et il a été approuvé par le *World Wide Web Consortium (W3C)* en février 1998.
XML n'est pas réellement un langage, c'est un métalangage à balise, utilisé pour définir des langages de description (XML-Schema, XUpdate, etc.).
Un document XML est composé de blocs de textes structurés par des balises de début et de fin, par exemple : <prenom>Ahmed</prenom>.

Un document XML peut non seulement contenir des données, mais aussi leur structure d'où le document transporte de l'information à propos des données qu'il contient. Ces données sont dans le format texte et ainsi les documents XML sont indépendants des plateformes.

Le fichier XML est structuré en éléments, les éléments peuvent contenir du texte et éventuellement d'autres éléments.
Le langage XML est extensible, c'est-à-dire, que l'on peut définir les noms de balises et la structure des balises librement dans un document XML. La seule contrainte à respecter est celle de la bonne formation. Pour qu'un document XML soit bien formé *(well formed document)* il doit respecter la syntaxe du langage XML définie par le W3C :

- Il existe un et un seul élément racine qui contient tous les autres éléments.

- Tout nom de balise doit commencer par une lettre ou un souligné (underscore) suivi, optionnellement, par des lettres, chiffres, traits d'union, points, deux points ou soulignés. La chaîne " xml ", quelle que soit sa casse, en début des noms est interdite (réservée pour de futurs usages).

- Toute balise ouverte doit être fermée, soit par = > si l'élément est vide, soit par la balise fermante correspondante (< =nomBalise >).

- Ces deux balises (ouvrante et fermante) doivent être correctement imbriquées entre les balises de l'élément parent (pas de recouvrement).

- Les attributs des balises, lorsqu'ils existent, doivent comporter obligatoirement une valeur qui doit toujours apparaître entre double apostrophe.

- Quand un élément est vide, les balises peuvent être simplifiées : <matricule> </ matricule > est identique à < matricule />

2.2. Document XML bien formé et valide

2.2.1 Document bien formé

Un document XML est dit bien formé s'il est syntaxiquement correct ; c'est-à-dire qu'il respecte les règles énoncées précédemment. Il est exigé que tout document XML à traiter soit bien formé.

2.2.2 Document valide

Un document XML est dit valide s'il est bien formé et conforme à la DTD ou au schéma XML *(XML schema)* qui définit sa structure. Cette condition est capitale dans la conception et l'implémentation de l'ETL parce qu'il faut veiller à ce que les informations importées dans l'entrepôt de données XML soient stockées en conformité avec les exigences du schéma XML qui sera utilisé par la suite pour déterminer les éléments de l'entrepôt de données.

2.3. Types de documents XML

Les documents XML peuvent se classer en deux grandes catégories : orientés données et orientés présentation.

2.3.1 Document orienté données

Les documents orientés données *(data-centric)* sont ceux ou XML est utilisé pour transporter des données. Leur structure physique (l'ordre des éléments, l'utilisation de PCDATA ou d'attributs) n'est pas importante dans la majorité des cas.
Ce sont des documents XML dont le contenu est fortement structuré avec des champs clairement sépares et bien identifies, par exemple la description des tables et des attributs dans une base de données relationnelle où le contenu des tables est inséré entre les balises.

2.3.2 Document orienté présentation

Les documents orientés présentation *(document-centric*) sont ceux dans lesquels XML est utilisé pour ses capacités similaires à SGML, comme dans un

manuel d'utilisateur, une page web statique en XHTML ou des brochures. Ils sont caractérisés par leur structure irrégulière et leur contenu mixte. A l'opposé des documents orientés données, leur structure physique est importante. Par exemple, pour un manuel d'utilisateur, l'ordre des chapitres est important. Par contre, pour une facture, l'ordre des articles ne l'est pas ou l'est moins. Ce sont des documents différencies des autres de par le fait que leur contenu est principalement composé de texte et non de champs.

Par son format simple, non-ambigu, indépendant de toute plate-forme et générique, XML facilite les échanges de données de types très différents entre des applications hétérogènes et les traitements par des outils standardisés.
On peut définir des règles pour la création de documents XML et dans ce cas on utilise les langages de schémas. Si un document XML X est conforme à un schéma, X est dit valide. Dans ce qui suit nous présentons les langages de schémas les plus utilisées à savoir les DTD et plus en détaille le XML schéma qu'on va utiliser par la suite comme langage de schéma dans notre démarche.

2.4. Les DTD et les Schémas XML

Un schéma de langage peut se baser sur une grammaire ou des assertions. Une grammaire contrôle les éléments autorisés dans le document XML, leur ordre, leurs occurrences, leur contenu ainsi que leur type de données tandis que les assertions contrôlent la nature des relations entre les éléments et les attributs dans un document XML.

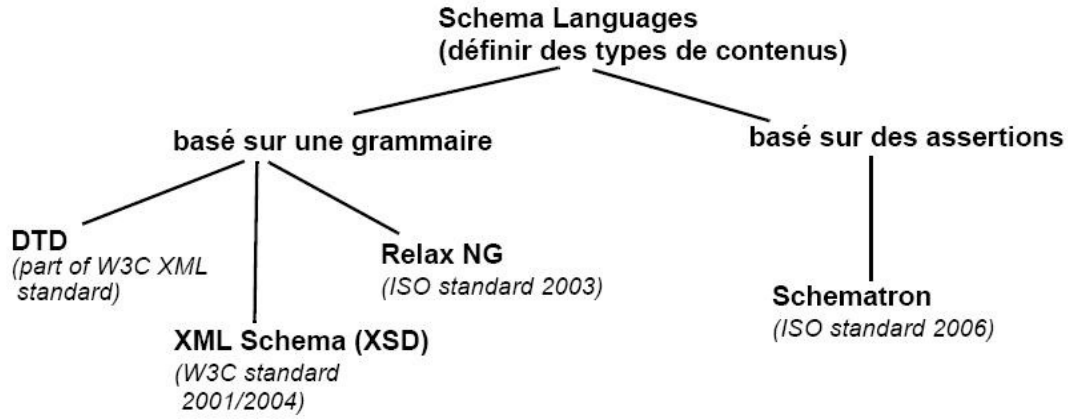


Figure 2.1 : Types de grammaires XML (Schémas de langages) [51].

On va s'intéresser aux DTD et aux schémas XML car ils sont les plus utilisés. Les DTD et les schémas XML sont des langages utilisés pour définir la structure des documents XML. Ils déterminent quels éléments peuvent être contenus dans un document XML, quels éléments peuvent être imbriqués dans d'autres, quelle valeur par défaut leurs attributs peuvent avoir, etc.
A l'aide d'une DTD ou d'un XML schéma et du document XML correspondant, un analyseur syntaxique peut confirmer que le document est conforme à la structure et aux contraintes désirées, un tel document est dit valide.

2.4.1 Les DTD

Une DTD est écrite selon la norme EBNF *(Extended Backus Naur Form)* d'ou elle est héritière du langage SGML.
Une DTD définit la structure du document à l'aide d'une liste d'éléments légaux. Le mot clé ! ELEMENT définit le type de l'élément et le mot-clé ! ATTLIST précise ses attributs.
La DTD permet également de déclarer le nombre de fois qu'un élément fils peut apparaître dans un élément parent : un ou plus (+), zéro ou plus (*) ou zéro ou un (?). Les attributs de type ID et IDREF permettent de créer des relations entre les attributs, à l'image des clés étrangères utilisées dans les systèmes relationnels.

Les types d'éléments possibles sont les suivants :

- #PCDATA désigne les données textuelles devant être traitées par l'analyseur.

- #CDATA désigne les données textuelles qui ne doivent pas être traitées par l'analyseur.

- EMPTY signifie que l'élément ne peut rien contenir.

- ANY signifie que l'élément peut contenir ce que l'on veut.

```
<?xml version="1.0" encoding="UTF-8"?>
<!ELEMENT rootelement (firstelement, secondelement)>
<!ELEMENT firstelement (level1)>
<!ATTLIST firstelement
position CDATA #REQUIRED>

<!ELEMENT level1 (#PCDATA | level2)*>
<!ATTLIST level1
children (0 | 1) #REQUIRED>
<!ATTLIST secondelement
position CDATA #REQUIRED>

<!ELEMENT level2 (#PCDATA)>
<!ELEMENT secondelement (level1)>
```

Figure 2.2 : Exemple de DTD: verysimplexml.dtd [52].

Bien que la DTD soit l'un des langages de schémas le plus utilisé, elle a des limitations, parmi lesquelles on peut énumérer :

- Une DTD définit très peu de types de données pour la validation du contenu et elles ont seulement le type texte (i.e., chaîne de caractères).

- Il est impossible d'intégrer d'autres définitions existantes puisque le support des namespaces différenciant les langages n'est pas supporté.

- Le langage n'est pas exprimé en XML, ce qui est en contradiction avec la forme même du langage défini.

Du fait des limitations énumérées ci-dessus, d'autres schémas ont été proposés tel que le schéma XML. On va présenter dans ce qui suit le schéma XML en détail vue qu'il sera utilisé par la suite comme point de départ de notre approche d'automatisation pour la conception d'un entrepôt de données.

2.4.2 Le schéma XML

Le schéma XML est un langage de description de format de document XML permettant de définir la structure et la grammaire d'un document XML. Il a fait l'objet de la recommandation du W3C datant en 2001.Le document produit à l'aide de ce langage est un fichier de description de structure (XML Schema Description ou fichier XSD). Ce dernier, au contraire de la DTD, est un document XML bien formé.

2.4.2.1 Les caractéristiques d'un schéma XML

- Il est écrit en utilisant le langage XML.

- Dans un schéma XML les éléments peuvent comporter des attributs et sont associés à un type de données qui peut être défini comme " type simple"(entiers, réels, chaine, date, liste…) ou " type complexe ". Un type simple est un élément ne contenant que du texte, tandis qu'un type complexe est un élément qui peut contenir d'autres éléments ou des attributs. Il est possible aussi de rendre un attribut optionnel ou obligatoire et de lui attribuer une valeur par défaut, c'est ce que l'on appelle les contraintes d'occurrences. Le schéma XML offre également la possibilité de constituer des " groupes d'attributs " afin de faciliter à la fois la lisibilité et la maintenance d'un schéma.

- Un schéma XML est constitué d'un ensemble de définitions de types et de déclarations d'éléments. Pour éviter tout conflit, chacun des noms qui leur sont respectivement attribués appartient à un espace de nom particulier appelé espace de nom cible (target namespace). Les espaces de nommage permettent donc de fournir un contexte à un vocabulaire, ce qui facilite la création de schémas et la validation de documents.
  <xsd :schema xmlns =" http : // www.w3.org /2001/ XMLSchema">
  (...) </ xsd :schema >

- Un schéma XML permet d'imposer des contraintes sur le nombre de fois qu'un élément peut apparaître dans le document instance (Contraintes d'occurrences). Cependant, contrairement aux DTD, la définition des contraintes dans un schéma XML se fait de façon beaucoup plus souple puisque tout nombre entier non négatif peut être utilisé pour fixer la valeur du nombre maximum et/ou minimum d'occurrences.

- Le schéma XML a été conçu de façon à supporter des mécanismes d'héritage (La dérivation de types). La dérivation par restriction et la dérivation par extension sont les deux techniques mises à disposition et s'appliquent aussi bien aux types de données simples que complexes. Cependant, afin d'éviter toute incohérence, le schéma XML permet de contrôler les dérivations effectuées. Ainsi, l'auteur d'un schéma peut préciser, pour un type simple ou complexe, s'il est dérivable ou non, et si oui, quel type de dérivation peut lui être affecté.

  - La dérivation par restriction : Cette technique permet de créer de nouveaux types simples, à partir de la bibliothèque de types simples intégrée à schéma XML. Ainsi, il est possible de restreindre l'ensemble des valeurs légales d'un type de base. Parmi les différentes façons qui existent, les notions " énumération " et "pattern " sont particulièrement utiles. La première permet d'énumérer l'ensemble des valeurs possibles pouvant être attribuées à un type simple, l'autre permet de définir des expressions régulières. Dans le cas de la restriction de types complexes, les valeurs représentées par le nouveau type sont un sous-ensemble des valeurs du type de base.

  - La dérivation par extension : L'extension de types simples consiste en fait, à créer des types complexes dont le contenu est un type simple pouvant porter des attributs. Pour ce qui est de l'extension de types complexes, elle permet de rajouter des éléments ou des attributs au modèle complexe de base.

Ci-dessous : Un exemple d'un document de schéma XML.

```
<?xml version="1.0" encoding="UTF-8"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema"
elementFormDefault="qualified">
    <xs:element name="firstelement">
        <xs:complexType>
            <xs:sequence>
                <xs:element ref="level1"/>
            </xs:sequence>
            <xs:attribute name="position" type="xs:boolean"
            use="required"/>
        </xs:complexType>
    </xs:element>
    <xs:element name="level1">
        <xs:complexType mixed="true">
            <xs:choice minOccurs="0" maxOccurs="unbounded">
            <xs:element ref="level2"/>
        </xs:choice>
        <xs:attribute name="children" use="required">
            <xs:simpleType>
                <xs:restriction base="xs:NMTOKEN">
                    <xs:enumeration value="0"/>
                    <xs:enumeration value="1"/>
                </xs:restriction>
            </xs:simpleType>
        </xs:attribute>
        </xs:complexType>
    </xs:element>
    <xs:element name="level2" type="xs:string"/>
    <xs:element name="rootelement">
        <xs:complexType>
            <xs:sequence>
                <xs:element ref="firstelement"/>
                <xs:element ref="secondelement"/>
            </xs:sequence>
        </xs:complexType>
    </xs:element>
    <xs:element name="secondelement">
        <xs:complexType>
            <xs:sequence>
                <xs:element ref="level1"/>
            </xs:sequence>
            <xs:attribute name="position" type="xs:byte"
            use="required"/>
        </xs:complexType>
    </xs:element></xs:schema>
```

Figure 2.3 : Exemple de Schéma XML : verysimplexml.xsd [52].

2.4.2.2 La structure d'un document schéma XML

- Espace de nom et préfixes

Comme tout fichier XML, un schéma XML commence par une déclaration XML et sa racine est la balise : <schema> ... </schema>.
Un schéma XML utilise des espaces de noms pour distinguer les éléments appartenant à XSD (le langage) et les éléments et les attributs définis par un schéma donné.

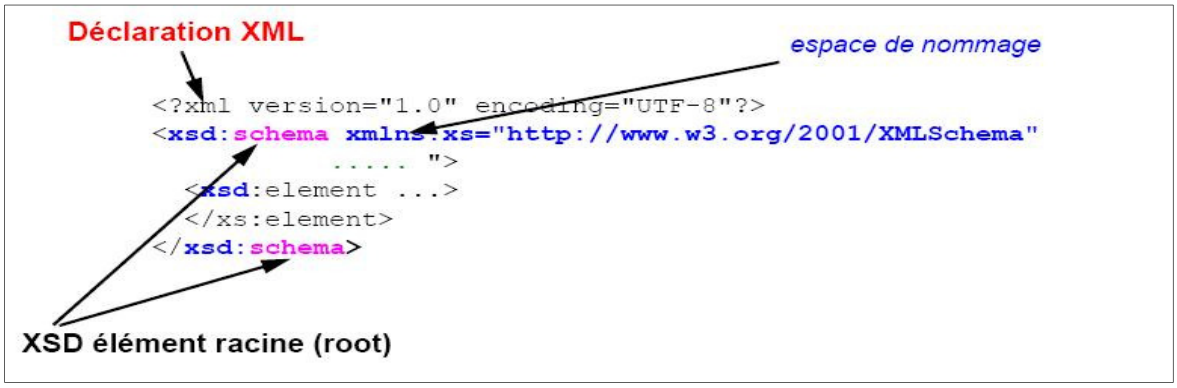


Figure 2.4 : La balise « schema » d'un schéma XML [51].

On peut soit définir un préfixe pour les éléments XSD soit pour nos éléments, on peut aussi choisir si nos éléments XML auront un namespace.
Dans la figure suivante, on montre comment donner un namespace aux éléments du schéma XML (xmlns:xs="http://www.w3.org/2001/XMLSchema "), les éléments XSD seront préfixés par xs : ou bien par xsd :, la balise elementFromDefault signifie que les balises n'auront pas de namespace.

```
<?xml version="1.0" encoding="UTF-8"?>
<!-- Simple recipe Schema -->
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema"
           elementFormDefault="qualified">
  <xs:element name="list">
    <xs:complexType>
      <xs:sequence>
        <xs:element maxOccurs="unbounded" ref="recipe"/>
      </xs:sequence>
    </xs:complexType>
  </xs:element>
```

Figure 2.5 : Donner un « namesapce» pour le code XSD [51].

Dans l'exemple suivant, les éléments définis pour le schéma ont un préfixe dans la définition alors que les éléments XSD ne seront pas préfixés.

```
<schema
  xmlns='http://www.w3.org/2000/10/XMLSchema'
  targetNamespace='http://yourdomain.org/namespace/'
  xmlns:t='http://yourdomain.org/namespace/'>

 <element name='t:list'>
  <complexType>
   <sequence>
    <element ref='t:recipe' maxOccurs='unbounded'/>
   </sequence>
  </complexType>
 </element>
```

Figure 2.6 : Donner un « namesapce » aux éléments du schéma [51].

- Validation

Pour associer un fichier XML à un schéma XML, on utilise l'attribut XSI.

```
<?xml version="1.0" encoding="ISO-8859-1" ?>
<list
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:noNamespaceSchemaLocation="recipe-no-ns.xsd">
  <recipe> ....
</list>
```

Figure 2.7 : Associer un fichier XML à un schéma XML [51].

- Définition d'éléments

Un élément peut être de type simple ou complexe, ci-dessous quelques exemples de déclarations d'éléments.

```
<xs:element name="author" type="xs:string"/>
```

Figure 2.8 : Définition d'un élément de type simple [51].

```
<xs:element name="recipe">
    <xs:complexType>
      <xs:sequence>
        <xs:element ref="meta"/>
        <xs:element minOccurs="0" ref="recipe_author"/>
        <xs:element ref="recipe_name"/>
        <xs:element ref="ingredients"/>
        <xs:element ref="directions"/>
      </xs:sequence>
    </xs:complexType>
  </xs:element>
```

Figure 2.9 : Définition d'un élément de type complexe [51].

```
<xs:element name="recipe" type="recipe_contents" />

<xs:complexType name="recipe_contents">
    <xs:sequence>
      <xs:element ref="meta"/>
      <xs:element minOccurs="0" ref="recipe_author"/>
      <xs:element ref="recipe_name"/>
      <xs:element ref="meal"/>
      <xs:element ref="ingredients"/>
      <xs:element ref="directions"/>
    </xs:sequence>
 </xs:complexType>
```

Figure 2.10 : Une autre définition d'un élément de type complexe [51].

- Les attributs

```
<xs:element name="Name">
 <xs:complexType>
   <xs:attribute name="lang" type="xs:string" use="required"/>
 </xs:complexType>
</xs:element>
```

Figure 2.11 : Définition d'un attribut pour un élément [51].

En résumé, le schéma XML est plus performant qu'une DTD car :

- Il est auto-descriptif, il utilise la syntaxe XML et il est compatible avec un grand nombre d'applications qui utilisent elles-mêmes XML.
- XML Schema permet de décrire l'imbrication et l'ordre d'apparition des éléments et de leurs attributs dans le but de définir une classe de documents XML, il offre la possibilité de contrôler l'existence d'éléments et d'attributs ainsi que leur contenu et autorise le typage de données spécifiques.

Caractéristiques	DTD	XML Schema (XSD)	Relax NG
Adoption	large	Applications data-centric	R&D, mais aussi qq. standards
Complexité structurale	moyenne	puissant (e.g. sets, element occurrence constraints)	puissant
Types de données	peu (10, surtout des valeurs d'attributs)	puissant (44 + plus types dérivables)	puissant
Complexité globale	basse	élevée	moyenne
Formalisme en XML	non	oui	oui (notation courte aussi)
Association avec un document XML	DOCTYPE declaration	Namespace declaration	Pas de solution standard
Support navigateur	IE (Firefox non)	non	non
File suffix	*.dtd	*.xsd	*.rng / *.rnc
Entités	oui	non (xinclude)	non

Tableau 2.1 : Tableau comparatif entre les schémas basés sur une grammaire [51].

2.5. Traitement de document XML

La récupération des données encapsulées dans le document nécessite un outil appelé analyseur syntaxique (*Parser*), permettant de parcourir le document XML et d'en extraire les informations qu'il contient. Les analyseurs XML sont également divisés selon l'approche qu'ils utilisent pour traiter le document. On distingue actuellement deux types d'approches : (1) Approche hiérarchique : les analyseurs utilisant cette technique construisent une structure hiérarchique contenant des objets représentant les éléments du document, et dont les méthodes permettent d'accéder aux propriétés. La principale API utilisant cette approche est DOM *(Document Object Model)*. (2) Approche événementielle, permettent de réagir à des événements (comme le début d'un élément, la fin d'un élément) et de renvoyer le résultat à l'application utilisant cette API. SAX (*Simple API for XML*) est la principale interface utilisant l'aspect événementiel.

2.5.1 DOM

(*Document Object Model*, traduisez modèle objet de document) est une spécification du W3C définissant la structure d'un document sous forme d'une hiérarchie d'objets, afin de simplifier l'accès aux éléments constitutifs du document, Cette hiérarchie, appelée DOM Structure Model ou modèle de structure DOM. Plus exactement DOM est un langage normalisé d'interface (*API, Application Programming Interface)* basée sur un arbre (Voir Figure 2.12), permettant à une application de parcourir la structure du document et d'agir dynamiquement sur celui-ci, Il est intéressant de constater que DOM est indépendant des langages de programmation et des plateformes.
La spécification DOM de niveau 1 définit un ensemble d'objets, dans le sens de la programmation orientée objet, qui peuvent représenter n'importe quel document structuré, comme un document XML.

La spécification de niveau 2 ajoute le support des dates et des espaces des noms. Le niveau 3 introduit des possibilités de chargement et de sauvegarde, ainsi que de validation. En effet, DOM est en général trop coûteux pour instancier des documents XML pendant le traitement. Il permet à un programme de manipuler le document XML en mémoire, où toutes les valeurs sont traitées comme des données textuelles de type DOMString.
Il accède via un parcours d'arbre aux contenues des éléments ou des attributs. Il est important de signaler que la taille utilisée par DOM et plus importantes que celle de documents XML.

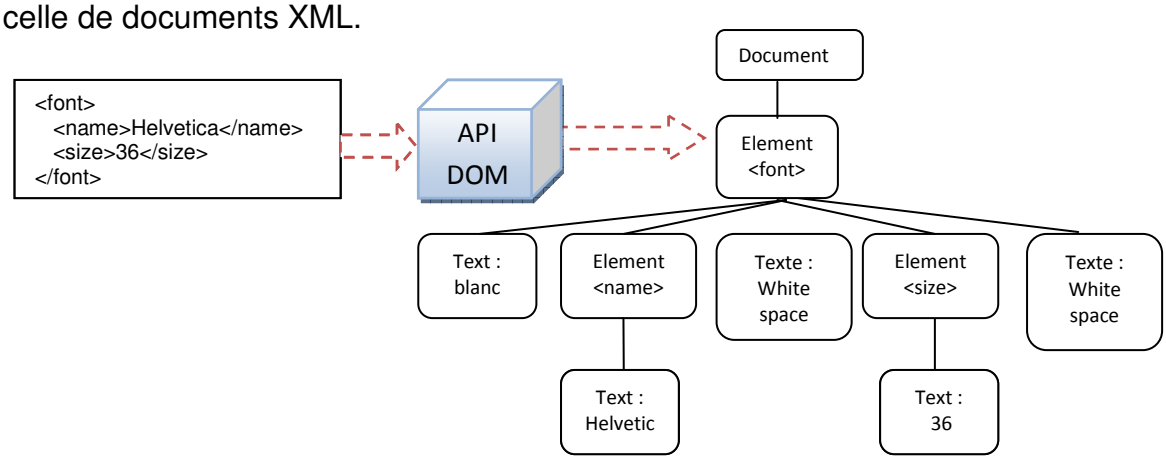


Figure 2.12 : Un exemple d'arbre DOM.

2.5.2 SAX

SAX est une API basée sur un modèle événementiel, cela signifie que SAX permet de déclencher des événements au cours de l'analyse du document XML, elle définit des triggers qui se déclenchent sur certaines balises. Adaptée aux applications qui extraient de l'information d'un document.
Une application utilisant SAX implémente généralement des gestionnaires d'événements, lui permettant d'effectuer des opérations selon le type d'élément rencontré. De nombreux analyseurs syntaxiques SAX sont disponibles comme SAXON.

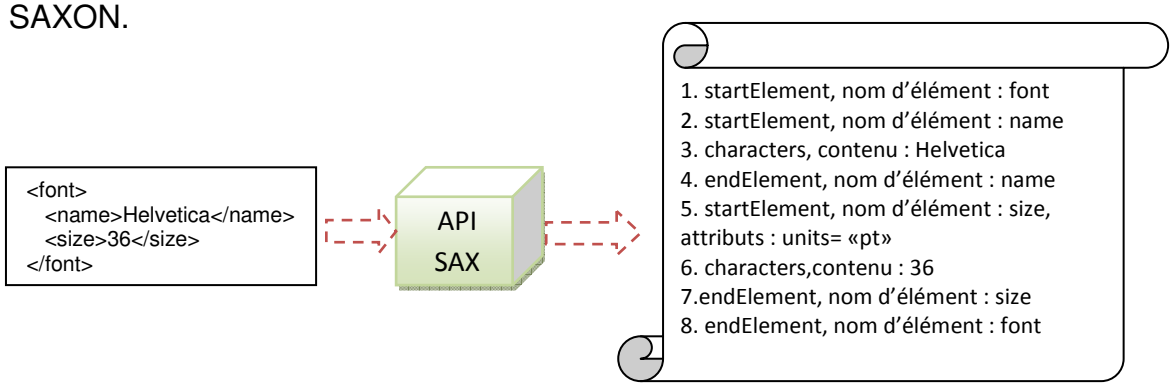


Figure 2.13 : Un exemple avec l'API SAX.

SAX permet de parcourir le document XML séquentiellement. A chaque nouvel élément et à chaque fin d'élément, un événement est déclenché.

Une application basée sur SAX peut gérer uniquement les éléments dont elle a besoin sans avoir à construire en mémoire une structure contenant l'intégralité du document. Dans le cas des entrepôts de données XML utilisant des SGBD XML natif, il est préférable de traiter les données XML sous forme de flux d'événement SAX pour des raisons d'efficacité, qui ne nécessite pas beaucoup d'espace mémoire contrairement à DOM.

Caractéristique	DOM	SAX
Modèle	Hiérarchique	Evénementiel
Structure en mémoire	Arbre d'objets	Objets événementiels
Chargement en mémoire	Tout le document	Seulement les éléments traités
Espace mémoire	Élevé	Faible

Tableau 2.2 : Tableau comparatif entre l'API DOM et SAX.

2.6. Le langage d'interrogation des documents XML

De nombreux travaux de recherche se sont intéressés à l'interrogation des données semi-structurées particulièrement les documents XML, on peut citer : Le SGMLQL **[53]** réalisant des requêtes sur des documents SGML, LOREL / OEM-QL **[54]** permettant la construction de graphes généralisés sur le modèle de données OEM, XPath **[55]** qui est un langage de requête simple pour exprimer des chemins dans un document. Plusieurs autres langages de requêtes plus complexes ont été proposés tel que XML-QL **[56]** (AT&T), XQL **[57]** (Microsoft), QUILT **[58]**, CDuce **[59]**, XyQL **[60]** (Xyleme/INRIA) et enfin XQuery **[61]** qui est devenu le standard du W3C pour l'interrogation de documents XML.

2.6.1 Lorel

Lorel **[54]** conçu comme un langage de requêtes pour des données semi structurées, puis a été étendu pour XML. Il est inspiré de SQL et de OQL. La syntaxe de base d'une requête Lorel est une expression de la forme :
select constructeur
from motif
where filtre

Un motif définit une relation en associant des variables à des sous-ensembles, comme en OQL, le langage des bases de données objets. Le constructeur crée un nouvel objet complexe pour chaque tuple de cette relation. La clause where est classique, il s'agit d'un filtre.

2.6.2 XPath

Le langage d'expression de chemin XML, XPath **[55]**, a été publié comme recommandation par le W3C en 1999. Jim Melton en fait une description complète dans son livre Querying XML. Selon sa spécification, XPath a été créé pour fournir une syntaxe commune pour les fonctionnalités partagées entre XSLT et XPointer. Son but est de pouvoir adresser des parties d'un document XML. Autrement dit, XPath permet de sélectionner un ensemble de nœuds de l'arbre XML à l'aide d'une expression de chemin.
/dblp/mastersthesis/title
/dblp//editor*
//title
//title/text()

2.6.3 XML-QL

Le langage est basé sur la reconnaissance de motif dans un graphe *(pattern matching)*, intègres tous les opérateurs relationnels pour l'extraction, la conversion et la transformation de données XML, il permet de formuler des requêtes sur des expressions de chemin. Une requête formulée dans ce langage contient deux parties, une partie qui est le document XML par des variables et une autre partie qui construit le résultat à l'aide des variables.

2.6.4 XQL

Est un langage proposé par Microsoft, il est similaire à XPath, il intègre des fonctionnalités de requêtes dans la syntaxe de XSL pour localiser et sélectionner des données dans le document XML, il fournit pour cela un ensemble d'opérations permettant de manipuler les données XML, en utilisant les même concepts de XPath comme '/' indiquant la racine de document, '//' indiquant les descendants de n'import quel niveau d'un élément, @ pour signaler un attribut. Il offre aussi la possibilité d'utiliser l'espace de noms (namespace).

2.6.5 XSLT

Extensible Stylesheet Language Transformations est un langage de transformation de documents XML créé en 1999 par le W3C. XSLT est conçu pour être utilisé comme une partie de XSL, le langage des feuilles de style de XML. En plus de XSLT, XSL inclus un vocabulaire XML pour la spécification de formatage. XSL spécifie les règles de présentation d'un document XML en utilisant XSLT pour décrire comment le document peut être transformé en un autre document qui utilise le vocabulaire de formatage.
XSLT est aussi conçu pour être utilisé indépendamment de XSL. Cependant, XSLT n'est pas censé être utilisé comme un langage de transformation XML à vocation générale.
Il a surtout été conçu pour les types de transformations nécessaires lorsque XSLT est utilisé comme une partie de XSL.

2.6.6 XQuery

XQuery est le langage aujourd'hui utilisé pour l'interrogation de documents XML. Il est issu de Quilt **[58]** proposé par IBM et certains auteurs de XML-QL.
Le premier document public de travail du W3C date du 15 février 2001. Il permet d'exprimer des sélections de chemins dans un document XML grâce à XPath. Chaque sélection est assignée à une variable définissant un ensemble d'arbres. Ces ensembles peuvent être manipulés par des contraintes, des ordonnancements, des constructions de résultat, des imbrications de requêtes, des quantificateurs, des agrégats, des séquences, des opérations ensemblistes et conditionnelles. Les spécifications du langage sont disponibles sur le site W3C **[61]**.

Il possède les caractéristiques suivantes :

- Composabilité : XQuery est un langage fonctionnel, ce qui veut dire que chaque expression XQuery retourne une valeur où lève une erreur. Ainsi les expressions XQuery peuvent être composées de façon assez libre, sous réserve de compatibilité de type.

- Fermeture : XQuery est fermé par rapport à son modèle de données, c'est-à –dire que le résultat de n'importe quelle expression XQuery forme une instance de son modèle de données et ne peut pas être en dehors de ce modèle.

- Conformité avec XML Schema : XQuery a été conçu de façon à être compatible avec XML Schema, c'est pour cela qu'il intègre plusieurs types provenant de XML Schema.

- Grand pouvoir d'expression : XQuery à un grand pouvoir d'expression, il, permet même d'exprimer la récursivité.

- Définition de fonctions : Dans une requête XQuery nous pouvons utiliser les fonctions définies dans une bibliothèque de fonctions décrites par la W3C comme nous pouvons définir nos propres fonctions.

XQuery est un langage fonctionnel dont chaque requête est une expression de différents types comme des expressions de chemin basées sur la syntaxe XPath, des expressions conditionnelles ou des fonctions et des expressions FLWR, cette dernière la plus important dans XQuery, peut se comparer à une expression type SQL select-from-where mais leurs fonctionnements respectifs sont assez différents. En effet, le principe est inversé d'un type d'expression à l'autre. Par exemple, en SQL, la construction des résultats se fait à l'aide de la clause select, au début de la requête. Tandis qu'en XQuery, elle se fait à l'aide de la clause return en fin de requête.

2.7. Conclusion

Dans ce chapitre, nous avons décrit la technologie XML et les outils de base qui lui sont associés, puis quelques langages d'interrogation XML.
Nous avons vu que XML permet une meilleure expressivité que la représentation relationnelle. Il s'agit bien d'une nouvelle technologie construite a priori pour échanger des données, mais qui a un impact beaucoup plus large, comme nous le verrons par la suite notamment dans le cadre des entrepôts de données.

CHAPITRE 3
L'ONTOLOGIE

3.1. Introduction

Le domaine de la représentation des connaissances a été pour longtemps un axe de recherche pour la communauté de l'intelligence artificielle **[62]**. La représentation des connaissances est un sujet multidisciplinaire qui applique les théories et les techniques de trois autres domaines : La logique, l'ontologie et la calculabilité **[63]**.
En réalité, La représentation des connaissances est l'application de la logique et de l'ontologie pour la construction de modèles de calcul pour certain domaines.
La logique fournit la structure formelle et les règles d'inférences, sans logique la représentation des connaissances n'a pas de sens clair. Une ontologie définie les types des objets dans un domaine d'application, sans ontologie les termes et les symboles sont indéfinis et confuses.
La calculabilité permet aux applications de distinguer entre la représentation des connaissances et la philosophie, sans les modèles de calculabilité, la logique et l'ontologie ne peuvent pas être implémenté dans des programmes informatiques **[63]**.
C'est dans ce contexte que s'articulent les travaux du mémoire que nous présentons, il s'agit précisément de proposer une représentation des connaissances pour automatiser la conception d'un entrepôt de données XML.

3.2. Quesque une ontologie?

Le terme « ontologie », un mot emprunté à la philosophie qui, s'il est d'origine grecque, ne fut manifestement créé qu'au XVIIe siècle. « Etude de l'être en tant qu'être, de l'être en soi » **[64]**, « Etude de l'existence en général, dans l'existentialisme » **[64]**, l'ontologie prend un tout autre sens en informatique, où le terme désigne un ensemble structuré de savoirs dans un domaine de connaissance particulier.
On distingue généralement deux entités globales au sein d'une ontologie. La première, à objectif terminologique, définit la nature des éléments qui composent le domaine de l'ontologie en question, un peu comme la définition d'une classe en programmation orientée objet, elle définit la nature des objets que l'on va manipuler par la suite. La seconde partie d'une ontologie explicite les relations entre plusieurs instances de ces classes définies dans la partie terminologique.
Ainsi, au sein d'une ontologie, les concepts sont définis les uns par rapport aux autres (modèle en graphe de l'organisation des connaissances), ce qui autorise un raisonnement et une manipulation de ces connaissances.

Un des moyens pour capturer des connaissances est l'utilisation des ontologies, une ontologie définie une forme concrète de la conceptualisation du domaine de connaissance d'une communauté **[65]**.
La définition traditionnelle d'une ontologie est 'L'ontologie est une spécification de la conceptualisation **[66]**, utilisée pour aider les programmes et les humains pour partager des connaissances' **[67]**. La conceptualisation est la capture des connaissances concernant le monde en termes d'entités(Les objets, les relations entre eux et les contraintes sur ces derniers).La spécification consiste à représenter cette conceptualisation sous une forme concrète **[65]**.Une des étapes de cette spécification est l'encodage de cette conceptualisation dans un langage de représentation des connaissances **[65]**.

Les ontologies peuvent avoir plusieurs usages **[67]** :

- Une référence pour une communauté : Elle permet la réutilisation des connaissances, une maintenance aisée et une mémoire de connaissances.

- Elle peut définir un schéma de base de données ou un vocabulaire commun pour une base de données d'annotations.

- Permettre un accès commun à l'information.

- Comprendre les annotations d'une base de données et les techniques de la littérature. Quelques ontologies sont conçue pour supporter le traitement du langage naturel (*NLP : Natural Language processing*), ces ontologies ne représentent pas seulement le domaine des connaissances mais elles permettent de savoir comment une connaissance est liée à une structure linguistique tel que une grammaire et à un lexique.

Les principales composantes d'une ontologie sont : les concepts, les relations, les instances et les axiomes.
Les concepts représentent un ensemble ou une classe d'entités d'un domaine et les relations décrivent les interactions entre les concepts ou les propriétés des concepts. Les instances sont les 'objets' d'un concept dans le domaine.
Il n'est pas nécessaire pour une ontologie d'avoir des instances car une ontologie est supposée être une conceptualisation d'un domaine, la combinaison d'une ontologie et ses instances est appelée une base de connaissances. Les axiomes sont utilisés pour mettre des contraintes sur les valeurs des classes ou des instances.
L'objectif principal d'une ontologie est la réutilisation **[68]**, ce qui la rend différente d'un schéma de base de données malgré que les deux soient des conceptualisations.

Généralement, une base de données est conçue pour satisfaire seulement une application tandis qu'une ontologie est destinée à être réutilisée par plusieurs applications. Les ontologies n'ont pas toutes le même objectif, il y a des ontologies qui ont des parties réutilisables et d'autres non réutilisables, ca varie selon leur couverture et le niveau de détail.

3.3. Construction d'ontologies

Le but de la construction des ontologies est la création d'un vocabulaire conventionnel et une structure sémantique pour l'échange d'information concernant le domaine **[69]**.
Il n'y a pas une méthodologie standard pour la construction d'ontologies, la méthodologie la plus connue a été développée par Gruber **[67]**, et aujourd'hui plusieurs recherches s'intéressent au développement de nouvelles méthodologies **[70]**.
Les méthodologies de développement d'ontologies peuvent se baser sur des phases telle que la méthodologie TOVE **[71]** ou sur l'amélioration itérative des prototypes telle que Methontology **[72]**. Les scientifiques font une distinction entre les techniques formelles et informelles de développement des ontologies, la définition informelle peut se faire en utilisant le langage naturel ou bien quelques diagrammes techniques tel que UML tandis que la définition formelle se fait en utilisant un langage de représentation des connaissances tel que OWL qui un langage compréhensible par la machine.

La figure 3.1 présente les phases communes à la majorité des cycles de vie de construction d'une ontologie.

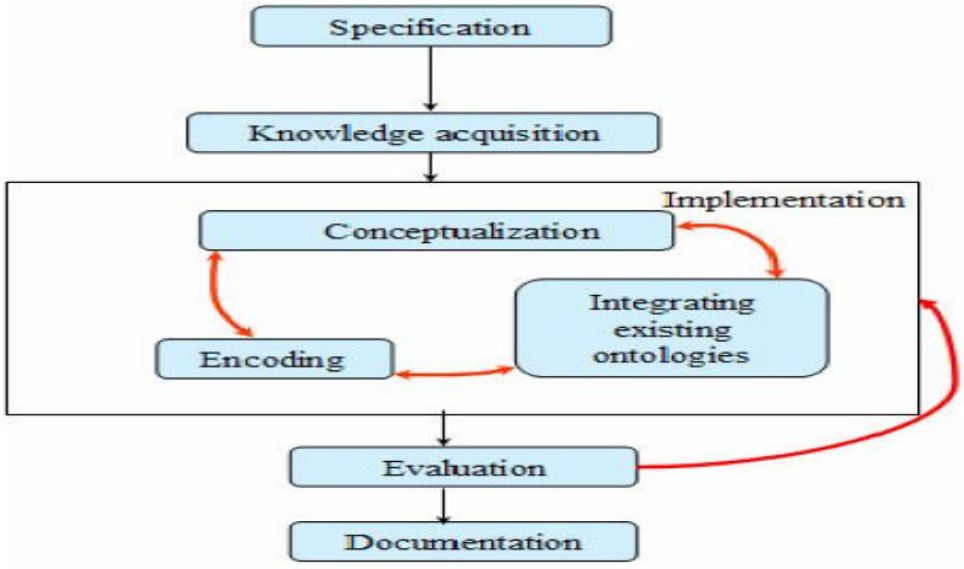


Figure 3.1 : Le cycle de développement d'une ontologie [69].

- La spécification : Cette phase définit le but, la couverture et la granularité du détail de l'ontologie. Elle est importante pour la conception, l'évaluation et la réutilisation de l'ontologie.

- L'acquisition des connaissances : Le processus consiste à récolter les connaissances depuis les spécialistes du domaine (Dans notre cas ces les concepteurs d'entrepôts de données), les métas données des bases de données, les livres, les papiers de recherches et d'autres ontologies.

- La conceptualisation : Elle identifie les concepts clés du domaine d'étude, leurs propriétés et les relations qui peuvent exister entre ces concepts.
  La détermination de ces concepts se fait sur la base des termes importants du domaine d'étude, ces termes sont spécifiés dans le langage naturel.
  Une fois ces concepts spécifiés, cette phase les structure en un modèle conceptuel.

- L'intégration : Elle réutilise ou combine les données des bases de données et des ontologies existantes dans le domaine d'étude pour construire une ontologie consistante.

- L'encodage : C'est la représentation de la conceptualisation dans un langage formel par exemple : Les frames, les modèles objets ou la logique.

- L'évaluation : Cette étape consiste à évaluer l'ontologie sur le degré de sa satisfaction aux besoins des applications tout en vérifiant l'aspect consistance, complétude et la non redondance des données **[73]**.

- La documentation : Une ontologie qui ne peut être comprise ne peut pas être réutilisée .Les définitions informelles et formelles, les commentaires et les exemples sont essentiels à rendre facile l'utilisation et la réutilisation de l'ontologie.

3.4. Langage de représentation des connaissances

A sa création par Tim Berners Lee, au début des années, l'objectif du web était de permettre des échanges rapides de savoirs entre individus distants. C'est dans ce but qu'a été créé le langage HTML, autorisant une mise en forme aisée et rapide documents en ligne.
Dix ans plus tard, au début du XXIème siècle, la « balkanisation » du Web a bien eu lieu. Tant au niveau de la présentation que du balisage ou du respect des standards, les documents présents sur le web ne constituent plus, de par leur hétérogénéité, une source de savoir fiable et agréable à consulter. Les connaissances que proposent les pages Web ont peu à peu été enfermées, au grès des modes, dans une couche présentation qui, si elle n'a pas dénaturé l'information, l'a en tout cas rendue moins accessible.
Depuis plusieurs années, toutefois, une nouvelle idée du Web prend corps : celle d'un Web Sémantique. Peu à peu, le W3C se dote de nouveaux langages: XML, RDF, OWL, n'en sont que des exemples. Tous ont pour objectif commun de participer à une formalisation des savoirs, en permettant ainsi un meilleur partage et une transmission plus aisée. Les champs d'application éventuels sont vastes : raisonnement automatique, résolution de problèmes par inférences, représentation de données structurées, traduction automatisée, etc.

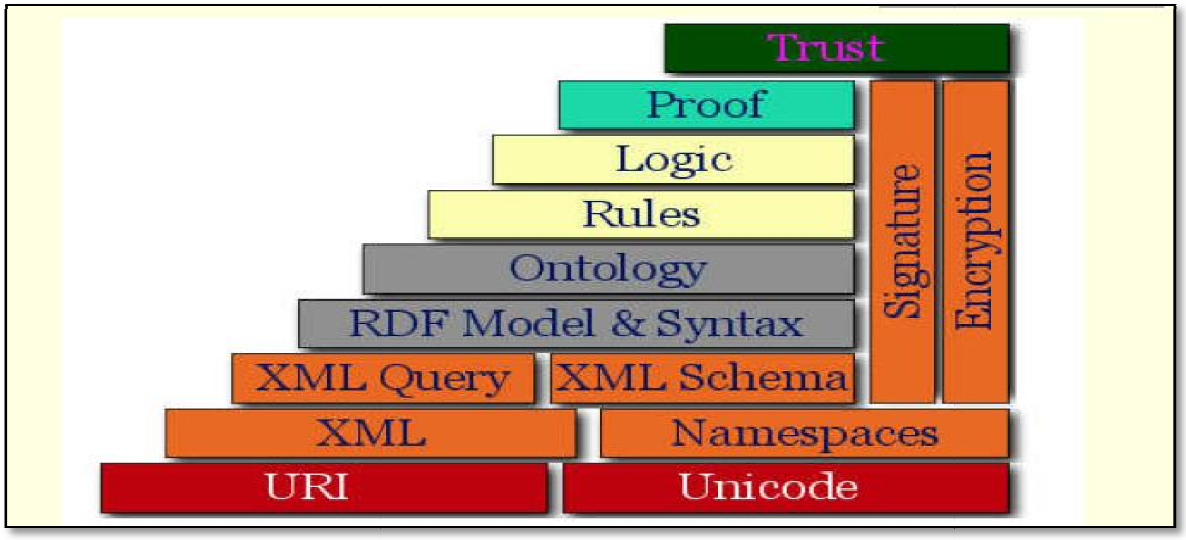


Figure 3.2 : Modèle en couches W3C [90].

Pour que les ontologies puissent être utilisées par une application, elles doivent être spécifiées et encodées dans un langage. Il y a une variété de langages qui peuvent être utilisés pour la représentation des modèles conceptuels, chacun des langages présente un degré d'expressivité, de facilité d'utilisation et de complexité de calculabilité.
Actuellement, il y a trois types de langages qui peuvent être utilisé pour spécifier une ontologie : les vocabulaires définis en utilisant le langage naturel, les langages orientés objet tel que les frames et UML, et les langages basés sur les prédicats qui sont exprimés avec la logique tel que la logique descriptive **[65]**.

Dans le tableau 3.1, nous présentons quelques langages utilisés dans la représentation des connaissances.

	KIF/OKBC/ CG/Cycl	**UML**	**TopicMaps /XTM**	**RDF(S)**	**DAML + OIL**	**OWL**
Description	Legacy KR Languages	Universal Modeling Language	Topic Maps /XML Topic Maps	Resource Discription Framework	DARPA ML + Ontology Inference	Web Ontology Language
Organisme	ANSI	OMG	ISO	W3C	DARPA	W3C
Nombre d'années depuis que le langage a été proposé	>6	>6	>6	>4	>3	>3
Open source	Oui	Oui	Oui	Oui	Oui	En cours

Tableau 3.1 : Les langages de la représentation des connaissances [74].

a) Langage basé sur le vocabulaire : C'est une description de l'ontologie fait à la main, elle consiste à décrire l'ontologie en utilisant la structure d'arbre. La position de chaque concept et ses relations sont complètements définis par le concepteur d'ontologie.

b) Langage basé sur les frames : Ce langage est basé sur la notion de frame ou de classe qui représente une collection d'objets. Chaque frame contient un ensemble de propriétés qui peuvent être remplie par des valeurs ou d'autres frames.
La communauté de la représentation des connaissances utilise beaucoup les systèmes basés sur les frames et spécialement dans les applications du traitement du langage naturel. EcoCyc **[75]** et RiboWeb **[76]** utilise la représentation frames.
La conception basé sur les frames est similaire à une conception orienté objets et elle est intuitive pour les utilisateurs, cependant elle présente quelque lacunes coté sémantique.

c) Langage basé sur la logique : Pour résoudre les problèmes sémantiques rencontrés avec le langage des frames, on peut utiliser la logique descriptive(DLs) **[77]**. DLs décrit la connaissance en termes de concepts et de relations qui sont déjà utilisés pour dériver une taxonomie de classification.

Le modèle décrit avec la logique descriptive est construit à partir de petites pièces d'une manière descriptive mais pas à travers d'assertion de hiérarchies.
Le moteur d'inférence DL associe à la logique descriptive permet d'inférer une classification d'hiérarchies et une vérification de la consistance des descriptions **[78]**.

3.5. Le langage web d'ontologie (OWL)

Le langage web d'ontologie OWL est conçu pour être utilisé par des applications qui ont besoin d'analyser le contenu des informations mais pas seulement de les présenter à l'utilisateur.
Récemment le W3C a mis une recommandation pour OWL, ce langage est utilisé pour représenter les ontologies. Dans les technologies du W3C on distingue aussi le langage XML qui offre une syntaxe pour les documents structurés mais sans aucune contrainte sémantique. Il y a aussi le RDF *(Resource Description Framework)* qui est un modèle de données pour les ressources et les relations entre ces dernières, ce modèle fournit une sémantique qui est représentée en XML. Le schéma RDF est un vocabulaire pour la description des propriétés et des classes.
Le langage OWL ajoute plus de vocabulaire **[79]** pour décrire les propriétés et les classes, il ne se limite pas aux relations entre les classes, il décrit les cardinalités, les équités, un typage riche des propriétés, les caractéristiques des propriétés et les classes énumératives.

On distingue trois sous langages OWL : OWL-Lite, OWL-DL et OWL-Full.

a) OWL-Lite : Il supporte une hiérarchie de classification et des contraintes simples, il est simple de fournir un outil qui supporte ce sous langage.

b) OWL-DL : Il supporte un maximum d'expressivité sans perdre la calculabilité de la complétude et de la décidabilité (Tous les calculs se terminent dans un temps déterminé) pour les systèmes de raisonnement **[79]**. OWL-DL correspond à une logique de description particulière nommée SHOIN(D) **[80]**. Ce sous-langage inclus touts les constructeurs OWL.
Nous allons utiliser OWL-DL pour concevoir notre ontologie vue ses caractéristiques (expressivité, calculabilité et décidabilité).

OWL DL Abstract Syntax	DL Syntax	Semantics
unionOf (C_1 C_2)	$C_1 \sqcup C_2$	$C_1^{\mathcal{I}} \cup C_2^{\mathcal{I}}$
intersectionOf (C_1 C_2)	$C_1 \sqcap C_2$	$C_1^{\mathcal{I}} \cap C_2^{\mathcal{I}}$
complementOf (C)	$\neg C$	$\Delta^{\mathcal{I}} \setminus C^{\mathcal{I}}$
restriction (P allValuesFrom(C))	$\forall P.C$	$\{x \mid \forall y.(x,y) \in P^{\mathcal{I}} \rightarrow y \in C^{\mathcal{I}}\}$
restriction (P someValuesFrom(C))	$\exists P.C$	$\{x \mid \forall y.(x,y) \in P^{\mathcal{I}} \wedge y \in C^{\mathcal{I}}\}$
restriction (P minCardinality(n))	$\geq n\ P$	$\{x \mid \|\{y \mid (x,y) \in P^{\mathcal{I}}\}\| \geq n\}$
restriction (P maxCardinality(n))	$\leq n\ P$	$\{x \mid \|\{y \mid (x,y) \in P^{\mathcal{I}}\}\| \leq n\}$
SubClassOf (C_1 C_2)	$C_1 \sqsubseteq C_2$	$C_1^{\mathcal{I}} \subseteq C_2^{\mathcal{I}}$
DisjointClasses (C_1 C_2)	$C_1 \sqcap C_2 \equiv \bot$	$C_1^{\mathcal{I}} \cap C_2^{\mathcal{I}} = \{\}$
EquivalentClasses (C_1 C_2)	$C_1 \equiv C_2$	$C_1^{\mathcal{I}} = C_2^{\mathcal{I}}$
ObjectProperty (P super(P_1))	$P \sqsubseteq P_1$	$P^{\mathcal{I}} \subseteq P_1^{\mathcal{I}}$
domain (C_1)	$\geq 1\ P \sqsubseteq C_1$	$P^{\mathcal{I}} \subseteq C_1^{\mathcal{I}} \times \Delta^{\mathcal{I}}$
range (C_2)	$\top \sqsubseteq \forall P.C_2$	$P^{\mathcal{I}} \subseteq \Delta^{\mathcal{I}} \times C_2^{\mathcal{I}}$
[inverseOf (P_2)]	$P \equiv P_2^{-}$	$P^{\mathcal{I}} = P_2^{\mathcal{I}-}$
[Symmetric]	$P \equiv P^{-}$	$P^{\mathcal{I}} = P^{\mathcal{I}-}$
[Functional])	$\top \sqsubseteq \leq 1 P$	$P^{\mathcal{I}}$ is functional

Figure 3.3 : La syntaxe et la sémantique abstraites OWL-DL [81].

c) OWL-Full : Il supporte un maximum d'expressivité et une liberté de description syntaxique du RDF se qui lui fait perdre sa garantie de calculabilité. OWL-full permet d'augmenter le sens du vocabulaire (RDF ou OWL) prédéfinie. Il n'y a pas de moteur d'inférence qui supporte tout les caractéristiques de l'OWL-Full **[79]**. Toute ontologie OWL-Full est une ontologie OWL-DL mais pas l'inverse.

3.6. OWL et les logiques descriptives

L'habilité d'utiliser les logiques descriptives rend le langage OWL un bon langage d'ontologie. Le raisonnement automatique en utilisant Racer supporte le processus de construction et d'évaluation d'ontologies.
Les logiques descriptives (DLs) ont plusieurs avantages dans la représentation des ontologies :

a) L'expressivité : DLs permettent une grande expressivité et une riche description du domaine des concepts.

b) Le raisonnement automatique : DLs sont basés sur des logiques formelles, ce qui permet le développement de moteurs d'inférence qui sont capable de vérifier la consistance des ontologies.

c) La composition : Permet d'avoir des ontologies qui couvrent un domaine plus large.

3.7. SWRL et SQWRL

Le SWRL est basé sur une combinaison des deux sous-langages OWL-DL et OWL-Lite avec les sous langages unaire et binaire Datalog et RuleML, il étend l'ensemble des axiomes OWL pour comprendre les règles Horn-like **[86]**.Ce langage utilise les règles de Horn-like qui ont la forme d'une implication entre un antécédent (body) et une conséquence (head), les conditions spécifiées dans la conséquence seront satisfaites une fois que les conditions de l'antécédent sont satisfaites. L'antécédent et la conséquence sont composés de zéro ou de plusieurs atomes. Un antécédent vide est évalué à vrai (Il est satisfait par n'importe quelle interprétation) d'où la conséquence est évaluée à vrai aussi par n'importe quelle interprétation d'autre part une conséquence vide est évaluée à faux (Elle n'est satisfaite par aucune interprétation) donc l'antécédent est aussi non satisfait par aucune interprétation. Plusieurs atomes sont traités comme une conjonction **[86]**. A noter qu'une règle avec une conjonction de conséquences peut être facilement transformée en un ensemble de règles chacune composée d'une conséquence atomique en utilisant les transformations de Lloyd-Topor **[87]**.
Les atomes de ces règles peuvent avoir la forme C(x), P(x,y), sameAs(x,y) ou differentFrom(x,y), ou C est une description OWL, P est une propriété OWL et x, y sont des variables, des individus OWL ou bien des valeurs de données OWL **[86]**.
La syntaxe d'une règle SWRL est : antécédent→ conséquence ou l'antécédent et la conséquence sont des conjonctions d'atomes. Les variables sont indiquées en utilisant une convention standard en les préfixant par un point d'interrogation (e.g., ?x).
SQWRL prononcé "squirrel" est un langage d'interrogation des ontologies OWL basé sur le SWRL. Il fournit les opérations SQL-like pour retrouver les connaissances dans une ontologie OWL **[91]**.

3.8. Pellet et RACER

Pellet est un moteur d'inférence pour le langage OWL DL, il est basé sur les algorithmes des tableaux développés pour la logique descriptive **[92]**. Il support tout le langage OWL DL en incluant les classes énumératives. Pellet est le seul moteur d'inférence qui permet de manipuler les constructions OWL owl :oneOf et owl :hasValue **[92]**. Il assure la sureté et la complétude en incorporant la récente procédure de décision développée pour le SHOIQ (the expressivity of OWL-DL plus qualified cardinality restrictions in DL terminology) **[92]**.

RACER *(Renamed ABox and Concept Expression Reasoner)* **[82]** est le premier système de logique descriptive. Il fournit un moteur d'inférence du web sémantique qui implémente TBox et ABox dans les logiques descriptives **[83]**.
Le moteur RACER est utilisé pour interroger des documents RDF et des ontologies OWL.
Le moteur RACER est très utilisé dans la communauté de la représentation des connaissances, beaucoup d'éditeurs d'ontologie supportent RACER.
RQL (Récemment nRQL (*new Racer Query Language)*) est un langage de requêtes pour le moteur d'inférence RACER. Le RQL peut être vus comme une extension et une combinaison de ABox (Des assertions sur des individus). Le RQL permet l'utilisation de variables avec les requêtes **[89]**.

3.9. Les outils de développement d'ontologie

Les outils sont essentiels au concepteur d'ontologies pour la construction d'ontologies et la fusion de plusieurs ontologies. Comme les modèles conceptuels sont souvent complexes, ces outils offrent des mécanismes pour visualiser et vérifier le modèle résultant.
Dans ce qui suit, nous présenter brièvement OiEd et Protege qui supportent tous les deux les moteurs d'inférences DL.

3.9.1 OiEd

OiEd **[84]** est un outil graphique pour la création et l'édition des ontologies OIL, il est développé à l'Université de Manchester. L'outil peut utiliser DAML+OIL et les langages OWL. OiEd utilise le système FaCT **[85]** qui est un système de logique descriptive pour vérifier les conséquences dans une ontologie. Le modèle de connaissances de l'outil est basé sur les logiques descriptives.

3.9.2 Protege

Protege a été développé par Stanford Medical Informatics à l'Université de Stanford. Il est destiné aux experts du domaine tel que les développeurs d'ontologies **[88]**.
Cet éditeur consiste en une interface graphique(GUI) qui est composé d'un ensemble d'onglets (Classes, instances, slots, formes et requêtes).

Cet outil offre les possibilités suivantes :

- Conception d'ontologie.
- Création d'un outil d'acquisition de connaissances pour collecter les connaissances.
- La saisie des instances de données et la création d'une base de connaissances(KB).
- L'exécution d'applications.

Au départ, Protege à été utilisé dans la médecine clinique et les sciences biomédicales, aujourd'hui il peut être utilisé pour modéliser une ontologie de n'importe quel domaine. L'outil supporte le langage OWL-DL.
Nous allons utiliser cet outil pour implémenter notre ontologie.

3.10. Conclusion

A travers ce chapitre, on a découvert le domaine de l'ontologie par son cycle de développement, ses langages de codage OWL et SWRL, son langage d'interrogation SQWRL, ses moteurs d'inférences et ses outils de développement (OiEd et Protege).

CHAPITRE 4
AUTOMATISATION DE LA CONCEPTION D'UN ENTREPOT DE DONNEES

4.1. Introduction

Nous allons présenter dans ce qui suit quelques approches proposées pour l'automatisation de la conception d'un entrepôt de données, nous nous sommes intéressés aux approches d'automatisation des entrepôts de données ordinaires ainsi que complexes (Données XML).
Nous commençons par présenter les systèmes d'analyse (OLAP) pour les données XML, puis nous détaillerons les différents travaux relatifs à l'automatisation de la conception d'un entrepôt de données, l'objectif principal d'étudier ces travaux est d'extraire les règles d'automatisation utilisées dans ces approches afin de les formaliser par la suite sous forme d'une representation des connaissances.
Dans le chapitre suivant nous proposerons cette representation des connaissances pour l'aide à la conception automatisée d'un entrepôt de données XML sous forme d'une ontologie.

4.2. OLAP et les données XML

Pour commencer, notons qu'il y a deux manières d'utiliser XML dans les systèmes d'analyse. La première consiste à utiliser XML pour représenter les données sources et/ou les résultats des requêtes. Tout le traitement des requêtes se fait à l'aide d'outils OLAP existants, basés sur des structures relationnelles ou multidimensionnelles. La seconde approche utilise XML aussi bien pour la représentation des données que pour leur traitement. Nous détaillerons ici ces deux approches.

a) Sources et résultats en XML : XML peut être utilisé pour représenter les données sources d'un système d'analyse. Dans cette approche, les informations intéressantes des documents XML sont extraites vers un moteur OLAP existant via un langage de traitement pour XML comme XSLT ou XQuery. L'analyse des données se fait à l'aide d'un moteur OLAP classique, basé sur des structures relationnelles ou multidimensionnelles.

b) Systèmes natifs XML : Cette approche utilise XML aussi bien pour la représentation des données que pour leur traitement. Les données sont

représentées par des arbres XML dans le moteur OLAP et sont traitées à l'aide de langages d'interrogation pour XML comme XQuery. Cette approche n'a pas encore été largement étudiée. En effet, très peu d'études sur le sujet ont été menées. Une de ces études, menée par Bordawekar chez IBM 2005 **[93]**, montre que le modèle multidimensionnel OLAP n'est pas adapté à l'analyse des documents XML et propose donc une approche pour analyser des documents XML en utilisant le modèle de données XML.

4.3. Entrepôt de données vs entrepôt de données XML

Dans le tableau suivant nous présentons une comparaison entre un entrepôt de données ordinaire et un entrepôt de données XML.

	Entrepôts de données	**Entrepôts de données XML**
Entrées	Données relationnelles	Fichiers XML, DTD et Schéma XML
Sorties	Cube (Dimensions, mesures et fait)	Cube (Un seul fichier XML regroupant les dimensions, mesures et fait ou bien plusieurs fichiers XML)
SGBD	Relationnel	XML native Relationnel
Interrogation	SQL	XQuery, XPath, XSLT

Tableau 4.1 : Comparaison entre un entrepôt de données et un entrepôt de données XML.

4.4. Approches de conception des entrepôts de données

Dans la littérature relative à la conception des entrepôts de données, il existe deux principales approches :
Les approches ascendantes « Bottom-up » qui sont guidées par le schéma de la source de données.
Les approches descendantes « Top-down » qui sont guidées par les besoins des utilisateurs décisionnels. A ces deux approches principales s'ajoute une approche mixte « Hybride », en réalité elle s'intéresse beaucoup plus soit aux sources de données ou aux besoins d'utilisateurs.

Le Tableau suivant résume la classification des travaux traitant le problème de conception et la construction des entrepôts de données selon le type de l'approche :

Auteurs	Type de l'approche	Propriété
Trujiilo et al, 2004 Vicky Nassis et al, 2004 Zhang et al, 2005 Rajugan et al, 2005 Boussaid et al, 2006 Tapio Niemi et al, 2007, 2009 Alberto Abelló et al, 2007	Orientée besoins d'utilisateur	Difficile à automatisée
Golfrelli et al, 2001 Pokorny 2001 Jensen et al, 2001 Vrdoljak et al, 2003 Hummer et al, 2003 Baril et al, 2003 Rusu et al, 2005 Li et An 2005 Park et al, 2005	Orientée schéma	Automatisable

Tableau 4.2 : Approches de conception des entrepôts de données.

4.5. Automatisation de la conception d'un entrepôt de données

Dans ce qui suit nous allons présenter en détaille des travaux relatifs à l'automatisation de la construction d'un entrepôt de données.

4.5.1 L'intégration des données OLAP en utilisant les ontologies et la technologie du web sémantique

4.5.1.1 Introduction

Usuellement les données utilisées dans l'OLAP sont limitées aux données provenant de l'entrepôt de données de l'entreprise. L'objectif de cet article **[95]** est de montrer qu'il est possible d'utiliser plusieurs sources de données en introduisant la technologie du web sémantique pour définir une ontologie OWL/RDF générale pour un cube OLAP et une autre pour le domaine d'étude qui intègre les différentes sources de données, la construction des instances de cube OLAP se fera par l'interrogation de ces ontologies par des requêtes RDF.

4.5.1.2 Description de l'approche proposée

Les méthodes de conception des systèmes OLAP, le recours aux manuels de travail pour la collection et l'intégration des données, les requêtes complexes et l'hétérogénéité des sources de données rendent le déploiement d'un système

OLAP une tache fastidieuse. Spécialement l'intégration d'un ensemble de sources de données différentes pose beaucoup de problèmes cela peut s'illustrer par l'exemple suivant : Dans deux schémas de base de données on peut trouver deux attributs qui ont le même nom mais sémantiquement ils sont différents. Une solution à ce problème consiste à définir une ontologie et des règles de passage des attributs locales vers les concepts de cette ontologie.

La méthode utilise une ontologie OLAP générique qui définie seulement les concepts généraux de l'OLAP. A cette ontologie s'ajoute une ontologie spécifique au domaine d'étude. Dans le cas ou on ne peut pas transformer une source de données dans l'ontologie spécifique au domaine on peut rajouter une autre ontologie qui définis les règles de mappage de ces données vers l'ontologie du domaine d'étude. Dans la figure 4.1, on présente l'ensemble des ontoogies utilisées pour integrer diveres sources de données ainsi que les téchnologies associées.

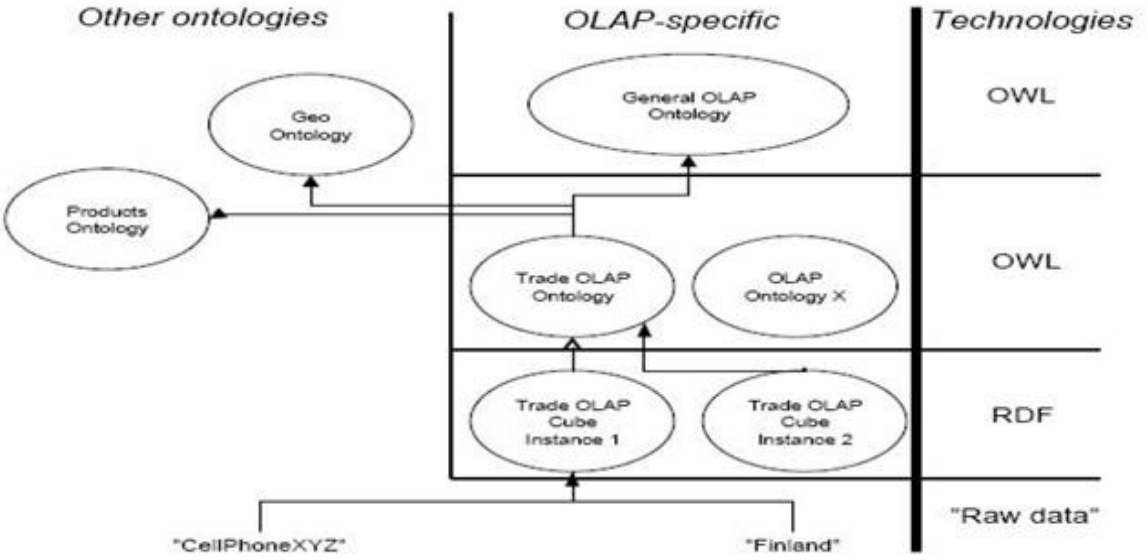


Figure 4.1 : Les niveaux d'ontologies et les technologies utilisées dans l'approche [95].

Dans le schéma de la figure 4.2, on montre les differentes transformations qui permettent d'integrer l'ensemble des sources de données hétérogenes pour la construction d'un cube OLAP.

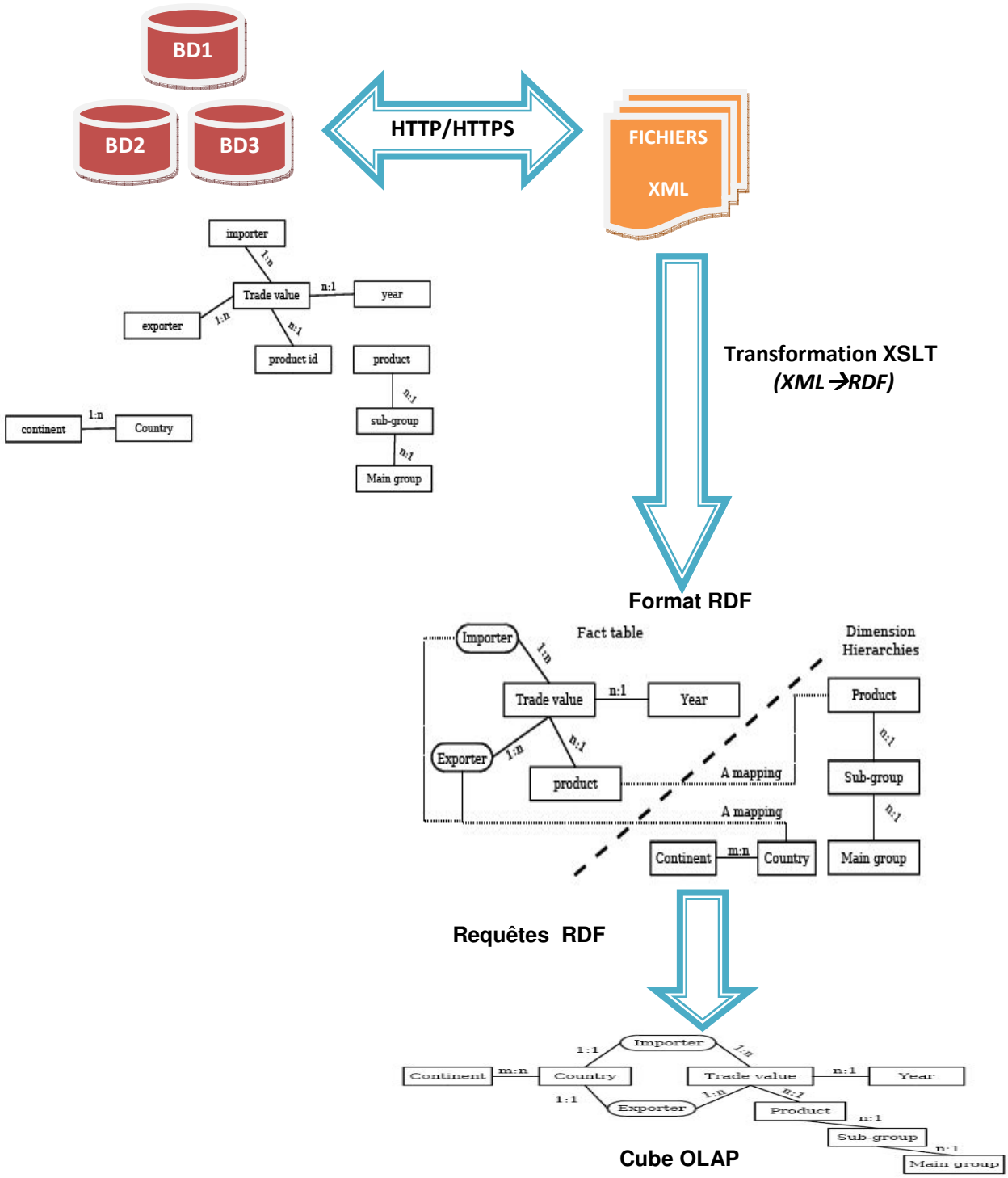


Figure 4.2 : Les étapes de l'approche proposée.

La démarche à suivre comporte les étapes suivantes **[95]** :

1. Transformer les données XML vers le format RDF en utilisant la technologie XSLT.

2. Interrogation des sources de données RDF en utilisant le framework Sesame.

3. Le calcul d'agrégation est fait en utilisant des scripts Perl.

4. Construction du cube OLAP à partir des données résultantes des requêtes en utilisant une application Java puis les charger dans un serveur OLAP avec un programme C.

5. Les commandes du serveur OLAP sont utilisées pour manipuler le cube OLAP.
   Les améliorations qu'apporte l'approche par rapport aux outils commerciaux et open sources sont :
   - Temps de repense réduit pour les requêtes.
   - Seulement les données pertinentes seront chargées dans le serveur OLAP.

a) L'ontologie globale

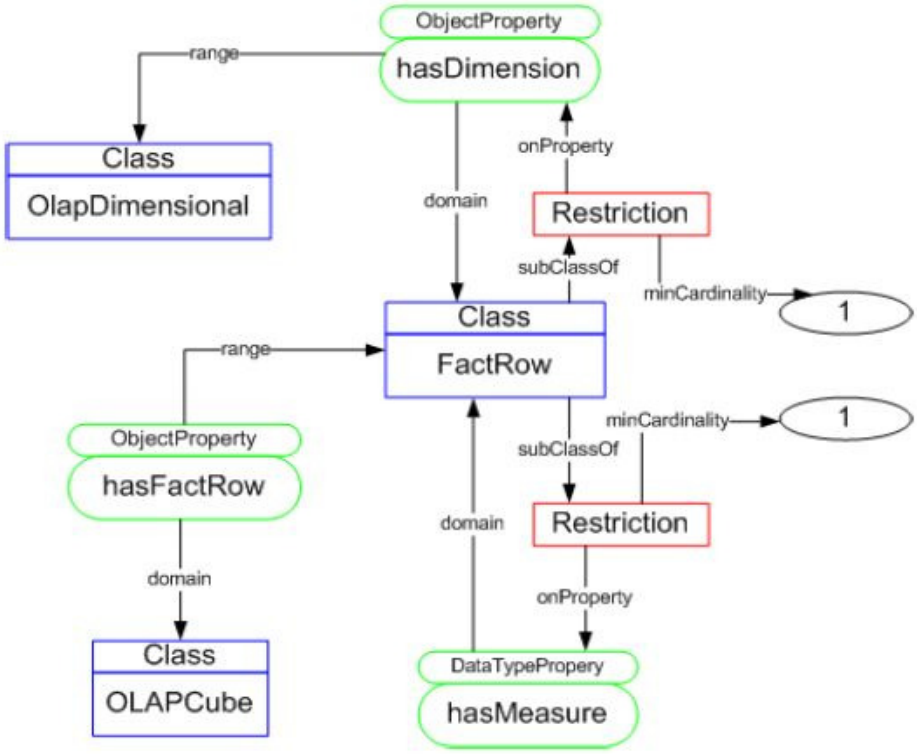


Figure 4.3 : L'ontologie OLAP [95].

b) L'ontologie spécifique au domaine

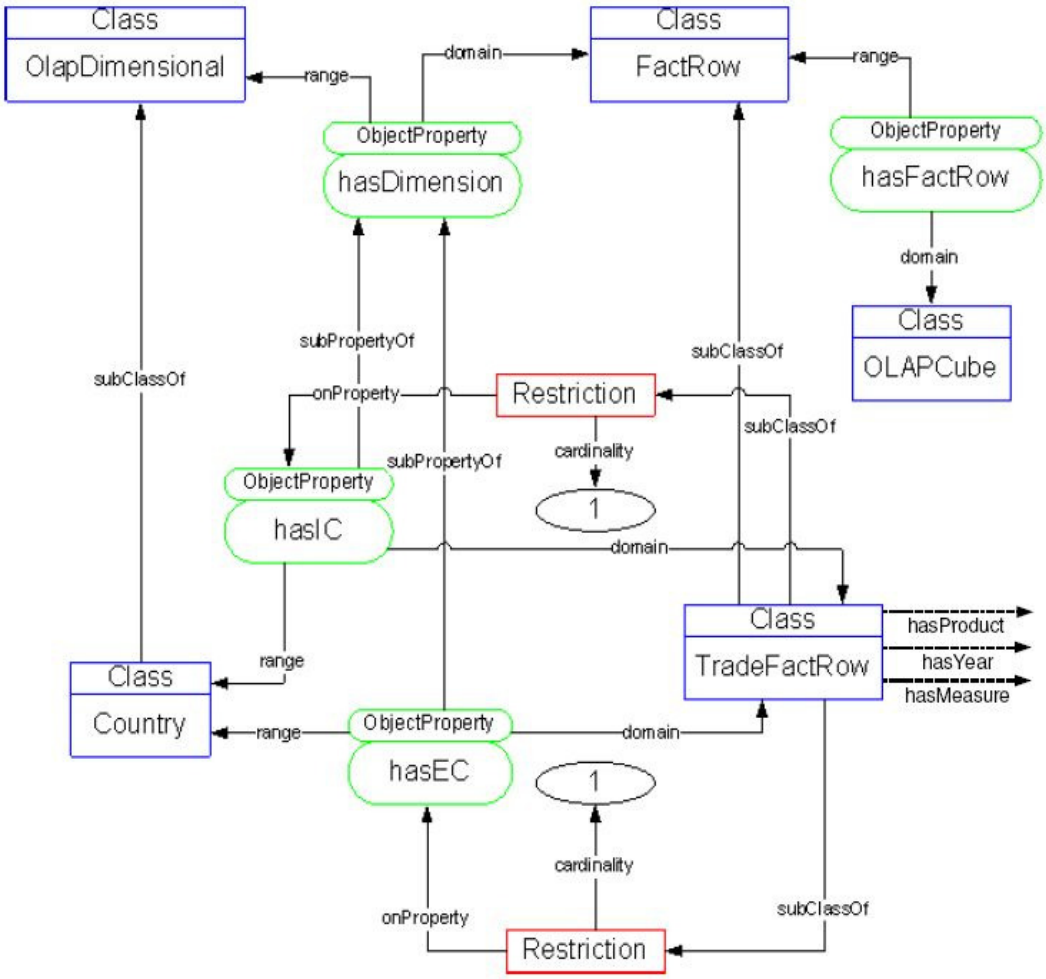


Figure 4.4 : L'ontologie Trade (Une partie de l'ontologie) [95].

4.5.1.3 Connaissances extraites

Ce qui nous intéresse dans ce papier c'est les concepts de l'ontologie OLAP décrite dans la figure 4.3, ci-dessous : l'ensemble des connaissances extraites.

- **C1.1 :** Un cube OLAP (OLAPCube) est composé d'un ensemble de lignes de fait (FactRow).
- **C1.2 :** Une ligne de fait (FactRow) est composé d'au moins une dimension (OLAPDimension) et d'au moins une mesure (OLAPMesure).

4.5.2 Le processus OLAP en utilisant les ontologies OWL et RDF

4.5.2.1 Introduction

Ce papier **[96]** présente une méthode de construction des cubes OLAP à la demande (*On-demand construction*) à partir des systèmes ROLAP. Cela veut dire que la construction d'un cube OLAP est orientée selon les besoins de l'utilisateur. L'approche est basée sur des ontologies RDF/OWL et des outils de conception.

Le peuplement d'un cube OLAP avec l'ensemble des données d'une base de données pose un problème d'espace et une inefficacité d'analyse. Cela peut s'avérer en pratique impossible si on utilise le web comme source de données. L'approche proposée essaye de résoudre ce problème ainsi que le problème d'hétérogénéité des sources de données.

La méthode utilise comme point de départ des sources de données hétérogènes et distribuées. Elle transforme ces données vers le format RDF en utilisant une ontologie de mappage, puis ces données RDF sont interrogées par des requêtes construites sur la base d'une ontologie OLAP. En finalité les données extraites sont stockées dans un serveur de base de données puis analysées avec des outils OLAP.

L'approche proposée utilise une ontologie OLAP générale **[100]** qui définie les concepts de dimensions et de mesures ainsi qu'une ontologie spécifique au domaine d'étude qui sera dérivée à partir de l'ontologie globale.L'auteur prend pour exemple deux ontologies : L'ontologie «Trade » qui décrit les ventes de divers produits entre différents pays et l'ontologie «World Fact Book» qui spécifie les caractéristiques des pays. La méthode essayera de répondre à des requêtes complexes telles que « Est ce que les pays ayant beaucoup de téléphones portables par 100 habitants exportent beaucoup de produits électroniques ? ».

L'auteur **[96]** présentera une méthode automatique pour la construction d'un cube OLAP, il spécifiera un ensemble de règles qui permet de manipuler un schéma OLAP en changeant ses dimensions et en finalité il donnera une preuve d'implémentation de ces concepts ainsi qu'un banc d'essai.

4.5.2.2 Présentation de la méthode

Le système proposé fonctionne comme suit :

a) Des données brutes et leurs ontologies sont utilisées comme entrées.

b) Ces données seront converties en fichiers RDF conforme à leur ontologie.

c) La construction d'un cube OLAP est faite en interrogeant les fichiers RDF avec des requêtes générées automatiquement en utilisant l'ontologie globale qui décrit un cube OLAP.

d) Le serveur Mondrian OLAP est utilisé pour stocker le cube OLAP.

e) L'utilisateur final va exploiter ce cube OLAP en utilisant une interface OLAP, il peut modifier la structure du cube OLAP.

L'auteur repose son approche sur le modèle relationnel et l'idée de la relation universelle. Il montre qu'une relation universelle peut être vue comme un cube OLAP non normalisé, cette relation représentée par des attributs renferme des dimensions et des mesures. Cependant il y a une combinaison de dimensions qui permet de déterminer une mesure, cet ensemble de dimensions représente les dimensions clés du cube OLAP. L'attribut dimension peut être soit un niveau d'une hiérarchie de dimensions ou bien un attribut qui apporte une information supplémentaire.

Le model OLAP est formalisé comme suit :

Definition1 : Le schéma et l'instance d'un cube OLAP

1. Un schéma de dimensions D et un ensemble de mesures M sont un ensemble d'attributs.

2. Un schéma de cube OLAP $C = D_1UD_2U...UD_nUM$, tel que D_1, …, D_n sont des dimensions, M est l'ensemble des mesures et $D_1UD_2...UD_n$ est la clé du cube C. Il existe un attribut clé unique K_k pour chaque schéma D_k.

3. L'attribut K_k est l'attribut dimension clé. Les autres attributs A € D_k sont les attributs dimensions.

4. Une instance de cube OLAP c est une instance de la relation C et une relation d sur D est une instance de dimension.

5. On suppose que l'ensemble des mesures n'est pas vide et un cube contient au moins une dimension d.

On se basant sur la definition1, l'auteur donne les dépendances fonctionnelles suivantes qui sont valides pour chaque cube OLAP :

Theoreme1 : Les dépendances fonctionnelles d'un cube OLAP

Chaque cube OLAP c ayant comme schéma $C = D_1UD_2U...UD_nUM$ respecte les dépendances suivantes :

1. L'ensemble des dimensions clés $K_1...K_n \rightarrow M$.

2. Chaque dimension clé $K_k \rightarrow D_k$.

On peut définir le niveau d'une dimension et l'attribut d'une dimension comme suit :

Definition2

Soit D un schéma de dimension. Un attribut A € D est un niveau de dimension s'il est utilisé comme niveau dans une hiérarchie de dimension sinon il est considéré comme un attribut d'une dimension.

Le modèle OLAP présenté ci-dessus est équivalent au modèle en étoile, l'auteur va présenter le modèle en étoile puis il va montrer l'équivalence avec le modèle OLAP proposé.

Definition3 : Le schéma en étoile et ses instances

1. Un schéma de dimension D_1, D_2,…, D_n et un ensemble de mesures M sont des ensembles d'attributs.

2. L'ensemble D_k est un ensemble de dimensions clés atomique tel que D'_1 est une clé atomique de D_1.

3. Une table de faits f est une relation sur le schéma D_1UD_nUM.

4. Une table de dimensions d_k est une relation sur un schéma de dimensions D_k.

5. Un schéma en étoile est un ensemble

$S = \{D_1UD_nUM, D_1, D_2, ...D_n\}$.

6. Une instance d'un schéma en étoile est un ensemble de relations {f, d1, d2, ...,dn} sur le schéma S.

7. On suppose qu'un schéma en étoile contient au moins un attribut mesure m € M et au moins une dimension clé D_1 et une instance d'une table de faits f est non vide.

Théorème2 : L'équivalence entre le modèle OLAP et le schéma en étoile

Le cube OLAP (Définition 1) est équivalent au modèle en étoile.

Preuve : Le schéma en étoile est équivalent au modèle OLAP de la définition1 quand la table des faits du modèle est normalisée.

Théorème3 : L'équivalence entre le modèle RDF/OWL et le schéma en étoile

Le cube OLAP de la définition3 est équivalent au modèle RDF/OWL.

Preuve : Chaque objet FactRow dans le modèle OLAP RDF/OWL à la forme (d_1, d_2,…, d_n, m_1, m_2, …, m_m) tel que d_1, d_2,…, d_n sont des dimensions clés et m_1, m_2, …, m_m sont des mesures, l'objet FactRow corresponds à une ligne de la table des faits dans une instance d'un cube OLAP. Les dimensions peuvent avoir des informations supplémentaires ce qui correspond aux tables dimension d'un schéma en étoile.

L'utilisateur va définir un schéma qui est un sous ensemble du schéma global en manipulant les dimensions de manière que ce schéma vérifie les règles 1 et 2 du théorème1. Les opérations qui peuvent être faites sur un cube sont :

1. Le changement des attributs non-clé dans une dimension.
2. Supprimer une dimension.
3. Changer une dimension.
4. Ajouter une dimension.

Lors du changement de la structure interne d'une dimension (En changeant les attributs non clés) il faut vérifier si la clé de la dimension détermine les autres attributs. Lorsqu'une clé de dimension est supprimée d'une table de faits, la règle1peut être non respecté de ce fait une redondance de clé dans la table des faits apparaît, ce problème peut être résolu en faisant des agrégations.

Voyant ça avec l'exemple suivant (Les colonnes(*) représentent les dimensions clés du cube OLAP) :

Product*	Product Group*	Exporter*	Value
Paper	Forest	FI	200
Wood	Forest	FI	250
Wood	Forest	SE	300
Phones	Electronics	FI	400

Après suppression de la dimension Product

Le cube OLAP

Exporter*	Value
FI	200
FI	250
SE	300
FI	400

(Redondance de la clé)

Après agrégation

(La somme)

Exporter*	Value
FI	850
SE	300

Définition4 : Une fonction d'agrégation

$$f(c) = \{(t[D], \sum_{t \in c'} t[M])\},$$

Equation 4.1 : La fonction d'agrégation [96].

Tel que :

- c : Est un cube OLAP.

- D : C'est l'ensemble des dimensions clés.

- M : Est l'ensemble de mesures.

- ∑ : Est une opération d'agrégation.

- c' : Est l'ensemble des tuples avec les mêmes instances de dimensions clés, c' = {t € c' s'il existe t' € c' tel que t [D] = t' [D]}.

Théorème4 : Suppression de la dimension clé

La suppression d'une dimension clé K d'un cube C peut se faire s'il existe une fonction d'agrégation f tel que chaque instance c' du schéma de cube C–K est un cube OLAP valide après l'application de f sur c'.

Théorème5 : Changement ou remplacement de dimensions

Une dimension clé K_1 atomique d'un schéma de dimension D peut être remplacer par un ensemble de dimensions clés atomiques $K'_1,\ldots,K'_n$, si la dépendance fonctionnelle suivante est vérifiée : $K'_1,\ldots, K'_n \rightarrow K_1$.

Théorème6 : Ajouter des dimensions

Soit c une instance de cube OLAP sur le schéma C et K' un attribut d'une dimension atomique D_k. Un cube OLAP c' sur le schéma C' = C U K' est aussi un cube OLAP valide.

4.5.2.3 L'architecture et l'implémentation de l'approche

La construction d'un cube OLAP suit deux phases :

a. La définition et la conversion des données

Pour chaque source de données (Quel soit une base de données relationnelle, un ensemble de fichiers ayant la structuration attribut=valeur ou bien un fichier XML) une ontologie de mappage est crée (La grammaire de cette ontologie est décrite par un fichier .XSD*)*, cette ontologie permet à partir de sources de données de produire des fichiers RDF conformes à l'ontologie OLAP.

Le processus de génération des fichiers RDF est décrit comme suit :

- Si les données se présentent sous forme de fichier attribut-valeur ou bien de tables SQL, une transformation vers le format XML (Qu'il soit un document D) est nécessaire.

- Pour chaque ontologie de mappage O, on évalue chaque expression « tuple/colonne » T comme une expression XPath pour D.

- Construire les fichiers RDF, s'il y a des transformations tels que la conversion de type de données alors il faut l'évaluer.

b. La création d'un schéma de cube et son peuplement

- La création d'un schéma Mondrian OLAP à partir de l'ontologie RDF/OWL (L'algorithme1).

- La création d'un schéma de base de données à partir du schéma Mondrian (L'algorithme2).

- Le peuplement de la base de données en utilisant les données RDF.

Algorithm 1 Create a Mondrian Schema from OWL OLAP schema.

1: Find the subclass C of the OLAPCube class.
2: Set Schema name = C and Cube name = C.
3: Find the dimension set CD related to C.
4: Set Table name = CD_fact.
5: **for all** dimension $D_n \in CD$ **do**
6: Create a <dimension> section
7: Set Dimension name = Dname, foreignKey = Dname
8: Set Hierarchy-AllmemberName = "Every "+D_n, primaryKey = D_n
9: Set Table name = Dname
10: **for all** level L in the hierarchy **do**
11: Set name = Lname, column = Lname
12: **end for**
13: **end for**
14: Find the measure set CM related to C.
15: Set name = CMname

Figure 4.5 : L'algorithme1 [96].

Algorithm 2 Create a relational database schema from a Mondrian OLAP Schema

1: **for all** subclass S of FactRow **do**
2: Identify DimensionSet D and MeasureSet M included in S
3: Create table D whose columns are the names of properties in D and M
4: **for all** class C in D **do**
5: Form a dimension hierarchy CH for C as explained above.
6: Create table C whose columns are the names of classes in CH.
7: **end for**
8: **end for**

Figure 4.6 : L'algorithme2 [96].

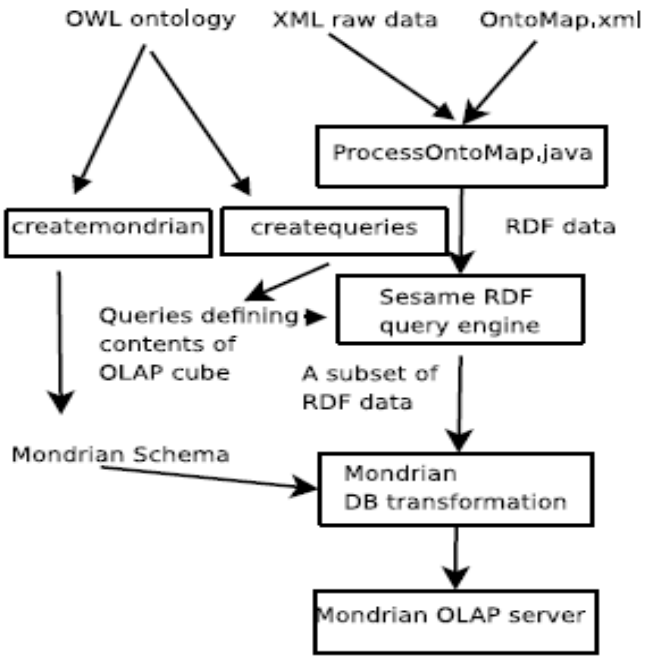


Figure 4.7 : Le workflow du processus [96].

4.5.2.4 Connaissances extraites

- Definition1 :
  - **C2.1 :** Un schéma de dimensions D et un ensemble de mesures M sont un ensemble d'attributs.

- D'après le théoreme 1(Les dépendances fonctionnelles d'un cube OLAP), on a chaque cube OLAP c ayant comme schéma C = $D_1UD_2U...UD_nUM$ respecte les dépendances suivantes :
  - **C2.2 :** L'ensemble des dimensions clés $K_1...K_n \rightarrow M$.
  - **C2.3 :** Chaque dimension clé $K_k \rightarrow D_k$.

- Théorème 2 :
  - **C2.4 :** L'équivalence entre le modèle OLAP et le schéma en étoile.

4.5.3. Une conception multidimensionnelle automatisée à partir d'ontologie

4.5.3.1 Introduction

On trouve dans la littérature plusieurs approches pour concevoir des entrepôts de données multidimensionnelles cependant elles sont manuelles, elles nécessitent l'intervention des experts. Ces dernières années quelques recherches ont essayé d'automatiser la conception des entrepôts de données multidimensionnelles mais ces approches se basent sur des modèles relationnels. L'approche proposée utilise comme entrée les ontologies car elles offrent un vocabulaire commun et en plus de ca l'ontologie est plus riche qu'un modèle relationnel.

Dans ce papier **[97],** l'auteur présente une méthode semi-automatisée qui consiste à utiliser des ontologies représentant des sources de données hétérogènes pour concevoir un entrepôt de données multidimensionnel.

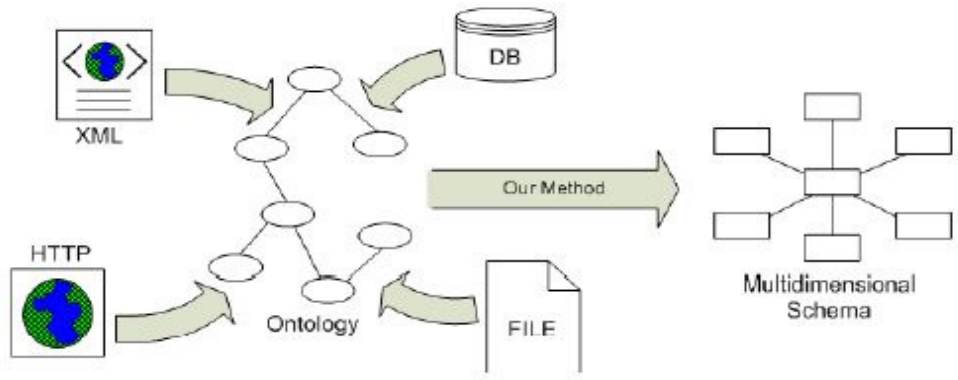


Figure 4.8 : La méthode proposée [97].

4.5.3.2 Présentation de la méthode

La méthode définie quatre critères qu'il faut respecter afin de construire le modèle multidimensionnel :

- [C1]Le modèle multidimensionnel : Un modèle multidimensionnel repose sur les concepts de fait et de dimension. Chaque dimension contient une hiérarchie de niveaux dotés de descripteurs et chaque fait contient des mesures. Un fait et plusieurs dimensions formeront le modèle multidimensionnel.

- [C2]La contrainte de disposition de l'espace multidimensionnel : La relation qui lie un fait à une dimension est de cardinalité 1 : N. Chaque instance de fait est liée à une et une seule instance des

dimensions analysées et une instance de dimension peut être liée à plusieurs instances de fait.

- [C3]La contrainte d'intégrité de la base : Les dimensions formant la base doivent être orthogonales, ils sont représentés par les clés primaires.

- [C4]La contrainte d'intégrité pour la synthèse : La synthèse des données doit respecter trois conditions (Ces conditions sont déduites intuitivement) :

  a) La disjonction (Les ensembles d'objets à agréger doivent être disjoints).

  b) La complétude (L'union des sous-ensembles doit constituer l'ensemble en entier).

  c) La compatibilité des dimensions (Quelques fonctions ne sont pas compatibles avec quelques dimensions et types de mesures, cependant la vérification de compatibilité reste manuelle dans l'approche proposée).

4.5.3.3 Description de l'approche proposée

La méthode proposée suit une démarche guidée par les besoins de l'utilisateur (Supply-driven méthode), l'utilisateur intervient pour restreindre le résultat de chaque étape.
La première tache consiste à chercher de potentiels faits, ensuite pour chaque fait elle va identifier sa base qui sera composée d'un ensemble de dimensions et à la fin la méthode va identifier d'éventuelle hiérarchie de dimension.

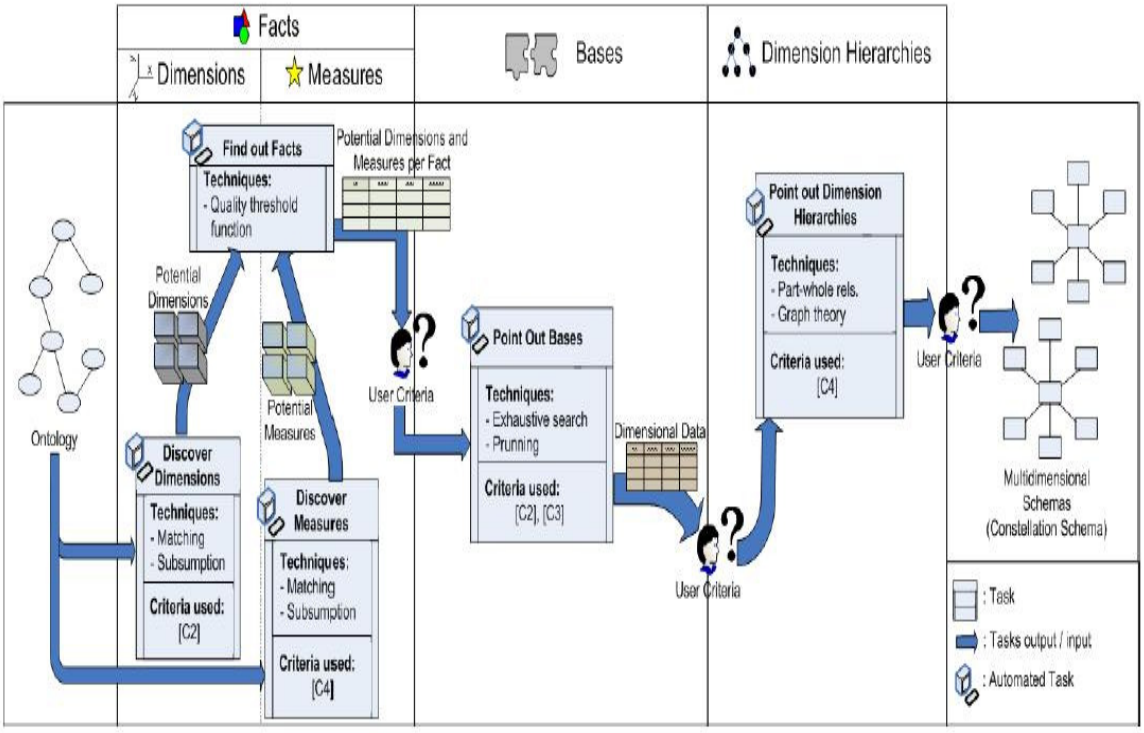


Figure 4.9 : Les étapes de la méthode [97].

4.5.3.4 Les détails de la méthode

Tache 1 : Détermination des faits

Un fait sera lié à un ensemble de dimensions et de mesures d'où il est nécessaire de trouver ces dimensions et ces mesures.

a- La découverte des dimensions

Pour chercher les dimensions, l'approche propose un algorithme fondé sur la relation mathématique suivante :

$$\mathcal{F} \sqsubseteq = 1r.\mathcal{D}, \quad where \quad r \equiv (r_1 \circ \ldots \circ r_n)$$

Equation 4.2 : Cette relation est exprimée en notation logique dont le langage OWL est basé [97].

En résumé cette écriture formelle dit que F et D sont deux classes qui représentent respectivement un fait et une dimension, chaque instance de D sera liée par une ou une composition de relations r à une et une seule instance de la classe F.

L'algorithme utilise une matrice de N*N concepts, cette matrice est représentée sous forme d'un vecteur de dimension N (N concepts) et chaque cellule en lui attache une liste d'éléments ayant la structure suivant (Concept, chemin composé d'un ensemble de relations de dépendances de cardinalité 1 du coté du concept destination).

- L'algorithme suit les étapes suivantes

1 - Il commence par initialiser le vecteur avec l'ensemble des concepts et met la liste rattachée à chaque concept à vide.

2 - Pour chaque concept il va chercher l'ensemble des concepts qui lui son directement rattaché avec une relation de cardinalité 1 dans le coté du concept destination, il mettra le couple (Concept trouvé, relation) dans sa liste.

3 - Tant que l'algorithme ne converge pas (Il y a des mises à jour de chemins), il va chercher les plus long chemins qui peuvent liés un concept du vecteur vers les autres concepts par composition de relation de cardinalité 1 du coté du concept destination.
La liste rattachée à chaque concept du vecteur contient de potentielles dimensions à ce dernier.

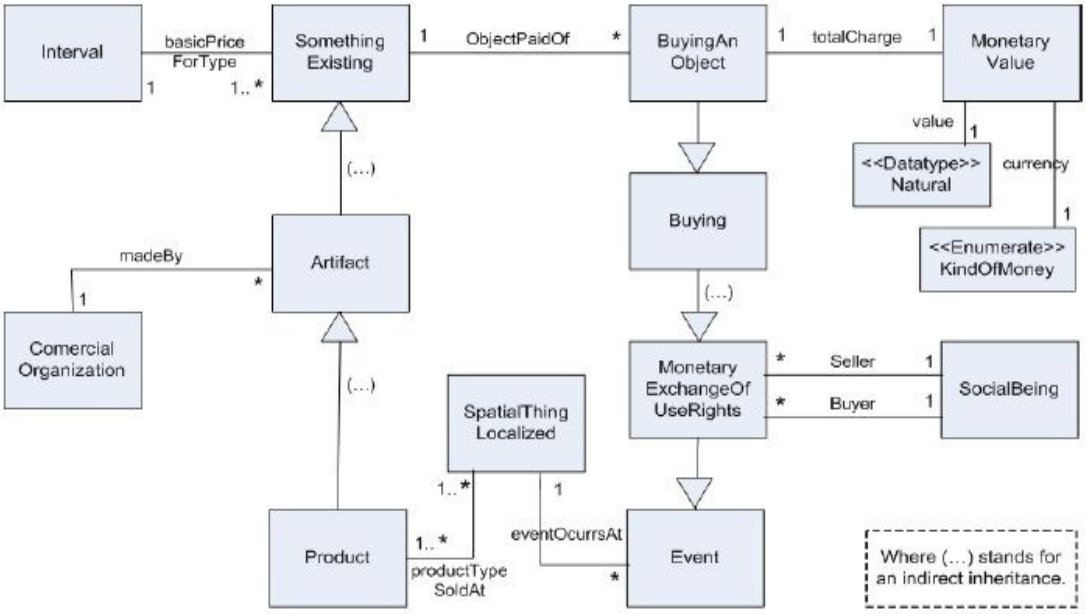
Enoncé de l'algorithme

```
typedef list <properties> path
typedef tuple < concept, list<path> > paths_to_concept
typedef tuple < concept, list<paths_to_concept> > func_depend
function create_matrix returns Matrix
   1. vector< list<func_depend> > M;
   2. bool converge = false;
   3. initialize(M);
   4. first_iteration(M, ontology);
   5. while(not converge)
        (a) propagate_path(M, converge);
   6. return M;
void propagate_path (Matrix ↕M, bool ↕converge)
   1. bool converge = true;
   2. foreach concept c in M do
        (a) func_depend ini_depend = M<c>;
        (b) foreach concept p in M<c> do
              i. M<c> ∪ (M<c<p>> ∘ M<p>);
        (c) if(ini_depend ! = M<c>) then
              i. converge = false;
```

Figure 4.10 : Algorithme pour chercher les dimensions des potentiels faits [97].

L'auteur prend comme exemple l'ontologie ResearchCyc décrite comme suit :

Après la première itération on aura le résultat suivant :



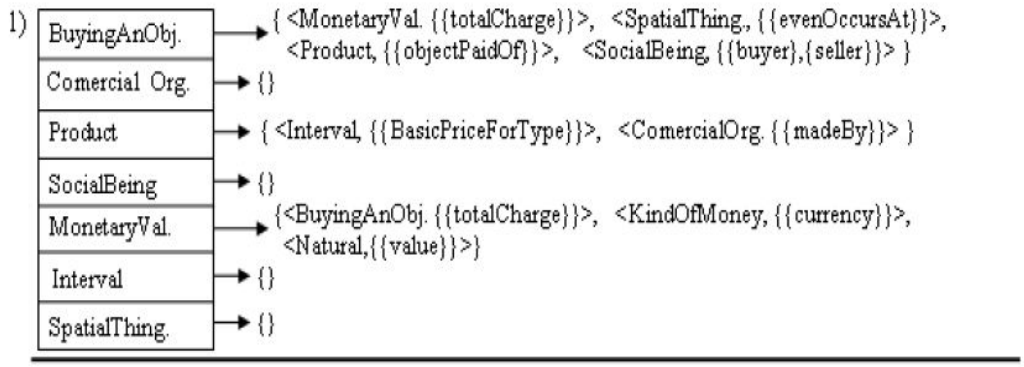


Figure 4.11 : Résultat de la première itération de l'algorithme qui recherche les dimensions [97].

Si on essaye de dérouler l'algorithme on ne peut pas déduire le résultat précédent (Première itération) a partir de l'ontologie décrite précédemment, en faite l'auteur supprime les relations de généralisation (Par définition dans UML un fils peut remplacer son père dans une relation de généralisation).

Deuxième itération :

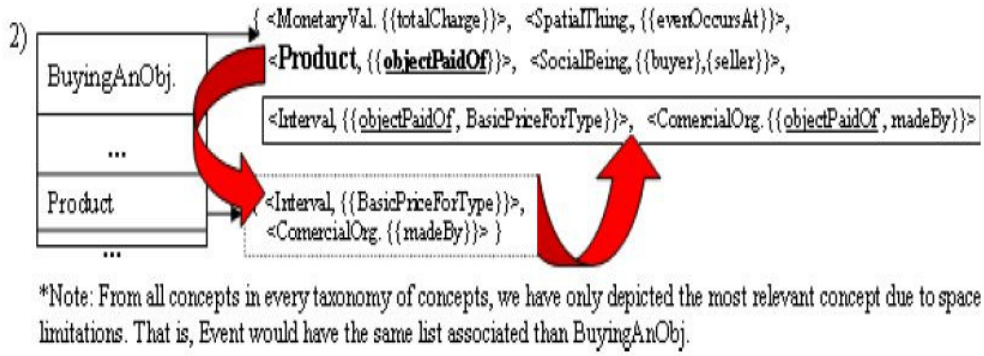


Figure 4.12 : Résultat de la deuxième itération de l'algorithme qui recherche les dimensions [97].

b- Découverte des mesures

Une mesure doit être un attribut numérique et elle doit respecter la contrainte C4 (Disjonction, complétude et compatibilité des dimensions).
L'algorithme qui cherche d'éventuelles mesures est le même que celui qui est décrit pour les dimensions sauf que la matrice contient des chemins de cardinalité 1:N du coté du concept mesure (Le concept source).

Multiplicity	Fact - Dim.	Fact - Measure
Min. Left	0	0
Max. Left	N	1
Min. Right	1	1
Max. Right	1	N

Figure 4.13 : Tableau résumant les multiplicités à respecter [97].

Concept	Dimensions	Measures
BuyingAnObj.	Interval, ComercialOrg., MonetaryVal, Value (datatype), Seller, Buyer, KindOfMoney, SpatialThing.	Value (Natural)
MonetaryVal.	Interval, ComercialOrg., BuyingAnObj, Value (datatype), Seller Buyer, KindOfMoney, SpatialThing.	Value (Natural)
Product	Interval, ConercialOrg.	-
Others	-	-

Figure 4.14 : Le résultat final de la tache «Détermination des faits» [97].

En conclusion, l'approche propose un ensemble de faits candidats à l'utilisateur en se basant sur une fonction de cout qui évalue le nombre de dimensions et de mesures rattachée à chaque concept (Dans l'exemple, les concepts BuyingAnObject et MonetaryVal sont proposés comme des faits candidats à l'utilisateur).

Tache2 : Chercher les bases

Cette étape va déterminer les bases (Pour rappel une base est un ensemble de dimensions orthogonales. La notion de base introduite dans l'article est identique à la notion de base définie dans la géométrie).

```
function seek_bases (Fact F, Matrix M) returns Set<Base>
1. int i=1; Set<Concept> Base;
2. Set<Base> Bases = {}, Combinations =
   Get_Potential_Dimensions(F,M), Candidates_Sets;
3. while(Combinations != Ø)
   (a) Candidates_Sets = {};
   (b) Base = Get_First_Combination(Combinations);
   (c) while(Base)
         i. if(Feasible_Base(Base)) then
              A. Bases += Base;
        ii. else if(Intermediate_Bases(Base)) then
              A. Candidates_Sets += Base;
       iii. Combinations -= Base;
        iv. Base = Get_Next_Combination(Combinations);
   (d) i++;
   (e) Combinations = Generate_Combinations_by_Size(i,
       Candidates_Sets, M);
4. return Bases;
```

Figure 4.15 : L'algorithme qui détermine les bases [97].

Tache3 : la détermination des hiérarchies de dimension

La phase de détermination de la hiérarchie de dimension reste une tache manuelle, elle nécessite l'intervention de l'utilisateur.

4.5.4 La conception d'entrepôt de données à partir de sources XML

4.5.4.1 Introduction

Dans ce papier **[98]**, l'auteur propose une approche semi-automatique pour construire un schéma conceptuel d'un entrepôt de données à partir de sources XML. On peut soit bâtir la conception sur les données XML ou bien sur les DTDs et les schémas XML. La méthode proposée consiste à translater les sources XML vers un schéma relationnel, puis concevoir l'entrepôt de données sur la base de ce modèle relationnel. Quelques travaux de recherche ont déjà étudié cette problématique sans s'intéresser beaucoup aux cardinalités des relations.

4.5.4.2 La modélisation et la conception multidimensionnelle

Plusieurs modèles conceptuels d'entrepôts de données existent dans la littérature, ils diffèrent par leur représentation graphique des concepts. L'approche utilisée se base sur le modèle dimension/Fait (*Dimensional Fact Model (DFM)).*

L'exemple étudié analyse le trafic d'un site web, il utilise le schéma fait/dimension CLICK décrit dans la Figure 4.16.

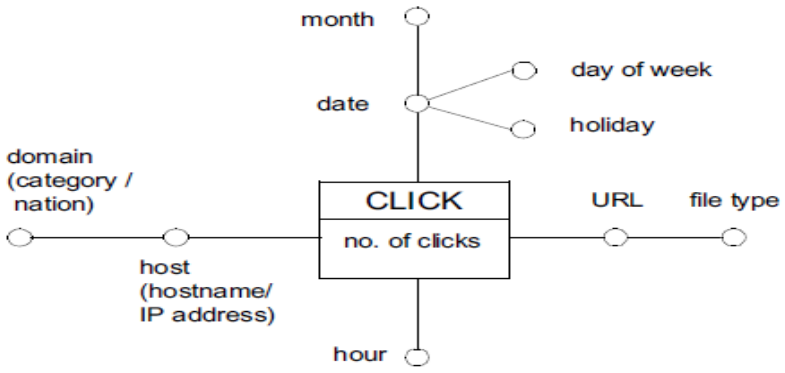


Figure 4.16 : Le schéma DFM montrant l'analyse du trafic d'un site web [98].

La plupart des approches de conception des entrepôts de données utilisent le schéma conceptuel ou logique des sources de données comme point de départ. L'élément commun à ces approches est la navigation à travers les dépendances fonctionnelles pour déterminer la hiérarchie d'un fait.

La démarche à suivre pour construire un DFM à partir d'un modèle E/R consiste à suivre les étapes suivantes:

- Choisir les faits.
- Pour chaque fait faire
  Construire l'arbre d'attributs.
  Ajuster l'arbre des attributs.
  Définir les dimensions et les mesures.

Fin pour.

Pour illustrer la méthodologie proposée, l'auteur va utiliser le diagramme E/R décrit dans la Figure 4.17.

En générale les faits correspondent aux événements qui se produisent dynamiquement dans le monde de l'entreprise, ils sont représentés par une entité F ou bien par une relation de cardinalité n aires, dans l'exemple l'entité fait est représenté par CLICK.

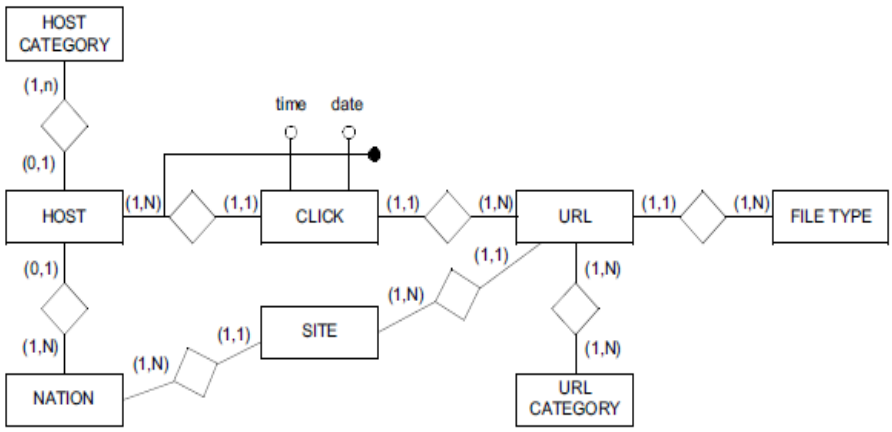


Figure 4.17 : Le schéma E/R de l'analyse du trafic d'un site web [98].

L'auteur essaye de construire un arbre d'attributs à partir du modèle E/R en suivant les étapes suivantes :

- Chaque sommet sera représenté par un attribut simple ou bien composé.
- L'entité F représentera la racine.
- Pour chaque sommet V de l'arbre d'attributs, l'attribut correspond détermine fonctionnellement tous les attributs descendants.

La construction de l'arbre d'attributs se base sur les dépendances fonctionnelles existantes dans le modèle E/R. Les relations père/fils entre les sommets sont déduites en navigant dans le modèle E/R à la recherche de toutes les relations de cardinalité 1:N du coté du fils et une cardinalité de 1:1 ou 0:1 du coté du père en démarrant de l'entité F sachant que l'attribut clé d'une entité sera représenté par un sommet et ses attributs non clé par des sommets fils à ce dernier.

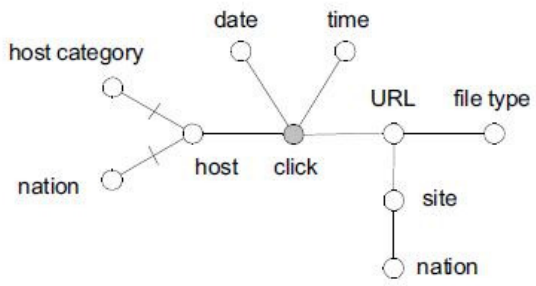


Figure 4.18 : L'arbre d'attributs du fait CLICK [98].

Une fois l'arbre d'attributs construit, il nous reste à déterminer les dimensions et les mesures parmi les fils de la racine fait CLICK, dans l'exemple présenté les dimensions sont : host, date, hour, et URL tandis que la mesure est le nombre de cliques d'un même utilisateur sur une même URL pour un instant donné (La date et l'heure du clique).Des arrangements peuvent être apportés sur l'arbre d'attributs en supprimant les sommets inutiles à l'analyse et en enrichissant d'autres. Dans l'exemple de l'article, l'auteur enrichi la dimension date par la construction d'une hiérarchie (month, day of week and holiday), en plus de ça les attributs host category et nation ont été remplacé par l'attribut domain.

4.5.4.3 La modélisation de relations dans XML

Un document XML est dit valide s'il respecte un schéma tel qu'une DTD ou un schéma XML. Comme la méthodologie proposée repose sur la recherche de relations de cardinalité 1:N de ce fait on va s'intéresser à savoir comment ces relations sont exprimées dans les schémas.

a) Les relations dans les DTDs

Une DTD définie la structure d'un document XML en utilisant les balises element-type pour déclarer les sous éléments d'un élément (Ses fils) et la balise attribute-list afin de définir les attributs (Leurs noms, types et valeurs possibles) d'un élément. En plus de ça il y a les mots clés ID qui définie un attribut comme une clé d'un élément, le mot IDREF pour référencier une clé étrangère ainsi que le mot IDREFS pour référencier plusieurs clés étrangères dans un document.

a.1) La modélisation des relations par la structuration sous-élément (sub-element)

Les relations dans une DTD peuvent être spécifiées par la structure de sous élément, la cardinalité de la relation est représentée par un caractère à la fin du nom de chaque élément fils. Les cardinalités que peut prendre une relation sont : 1:N représentée par +, 0:N représentée par *, 0:1 représentée par ? et 1:1 la cardinalité par défaut.

La structuration des éléments de la DTD avec la relation sous élément ne peut représenter que des cardinalités 1:1 ou 1:N et on ne peut spécifier qu'une seule direction dans une relation (Dans l'exemple de la figure 4.19, on remarque que la

relation entre l'élément URL et l'élément urlCategory n'est spécifiée dans la DTD que dans un seul sens). En plus de ça lors de la définition d'une clé étrangère, on ne peut pas spécifiée l'unicité du type de données de cette dernière.

```
<!DOCTYPE webTraffic [
   <!ELEMENT webTraffic (click*)>
   <!ELEMENT click (host, date, time, url)>
   <!ELEMENT host (category | nation)>
   <!ATTLIST host
      hostId ID #REQUIRED>
   <!ELEMENT category (#PCDATA)>
   <!ELEMENT date (#PCDATA)>
   <!ELEMENT time (#PCDATA)>
   <!ELEMENT url (site, fileType,
urlCategory+)>
   <!ATTLIST url
      urlId ID #REQUIRED>
   <!ELEMENT site (nation)>
   <!ATTLIST site
      siteId ID #REQUIRED>
   <!ELEMENT nation (#PCDATA)>
   <!ELEMENT fileType (#PCDATA)>
   <!ELEMENT urlCategory (#PCDATA)>
]>
```

Figure 4.19 : La DTD (Les relations sont spécifiées avec la structure sous–élément) [98].

a.2) La modélisation des relations par ID et IDREF(S)

On peut spécifier les relations entre les éléments dans une DTD en utilisant les contraintes ID et IDREF(S), ces contraintes sont semblables aux mécanismes de clé primaire et de clé étrangère utilisés dans les bases de données relationnelles avec quelques limitations.

```
<!ELEMENT webTraffic (click*, fileType+,
urlCategory+)>

<!ELEMENT url (site)
<!ATTLIST url
   urlId ID #REQUIRED
   fileTypeRef   IDREF #REQUIRED
   urlCategoryRef IDREFS #REQUIRED>
<!ELEMENT fileType EMPTY>
<!ATTLIST fileType
   typeID ID #REQUIRED
   typeDescription CDATA #IMPLIED>
<!ELEMENT urlCategory EMPTY>
<!ATTLIST urlCategory
   urlCategoryId ID #REQUIRED
   urlCategoryDesc CDATA #IMPLIED>
```

Figure 4.20 : La DTD en spécifiant les relations avec IDREF(S) [98].

b) Les relations dans les schémas XML

Le schéma XML permet une représentation plus expressive qu'une DTD, en particulier les cardinalités avec l'utilisation des contraintes minOccurs et maxOccurs. Le schéma XML utilise les mots clés key et keyRef pour spécifier une clé primaire et une clé étrangère.

b.1) La modélisation des relations par la structuration sous - élément

Cette représentation est semblable à celle d'une DTD structurée en sous élément décrite dans la section a.1 sauf qu'on utilise les mots clés minOccurs et maxOccurs.

```
<element name= click >
   <complexType>
      <sequence>
         <element name= host
                  type= hostType />
         <element name= date  type= date />
         <element name= time  type= time />
         <element name= url  type= urlType />
      </sequence>
   </complexType>
</element>

<complexType name= urlType >
   <sequence>
      <element name= site  type= siteType />
      <element name= fileType
               type= string />
      <element name= urlCategory
               type= string
               maxOccurs= unbounded />
   </sequence>
   <attribute name= urlID  type= string />
</complexType>
```

Figure 4.21 : Un schéma XML en utilisant la structuration sous élément [98].

b.2) La modélisation des relations par les éléments key et keyRef

En plus des mots clés ID et IDREF(S), le schéma XML offre plus de possibilité de représentation avec les contraintes key et keyRef. Le mot clé key permet d'indiquer qu'un attribut ou un élément est unique non nul et le mot keyRef offre la possibilité de référencier d'autres clés. Le schéma XML permet de spécifier la portée (Scope) de chaque clé en utilisant une expression XPath.

```
<fileTypes>
   <type typeID= html >
         hypertext markup language</type>
   <type typeID= gif >
         graphic interchange format</type>

</fileTypes>
```

```
<element name= click >
   <complexType>

      <element name= url  type= urlType />
      <element name= fileTypes
                type= fileTypesType />

   </complexType>
   <key name= fileTypeKey >
      <selector xpath= fileTypes/type />
      <field xpath= @typeID />
   </key>
   <keyref name= fileTypeRef
           refer= fileTypeKey >
      <selector xpath= url />
      <field xpath= fileType />
   </keyref>
</element>
```

Figure 4.22 : Un schéma XML en utilisant key et keyRef [98].

4.5.4.4 Une conception à partir de sources XML

Dans cette section, l'auteur présente une approche semi-automatique pour la construction d'un schéma conceptuel d'un data mart à partir de données XML. Il considère que les données XML sont conformes à une DTD utilisant la relation de sous-élément.

La méthodologie suit les étapes suivantes :

a. Simplifier la DTD.

b. Créer le graphe DTD.

c. Choisir les faits.

d. Pour chaque fait

d.1 Construire l'arbre d'attributs à partir du graphe DTD.

d.2 Arranger l'arbre d'attributs.

d.3 Définir les dimensions et les mesures.

a. Simplifier la DTD

Les sous éléments ayant le même nom seront groupés, les opérateurs unaires seront réduits à un opérateur unaire et les opérateurs « + » seront transformés en « * » (Par exemple host (catgory/nation) sera transformé en host (category ?,nation ?).

b. La création du graphe DTD

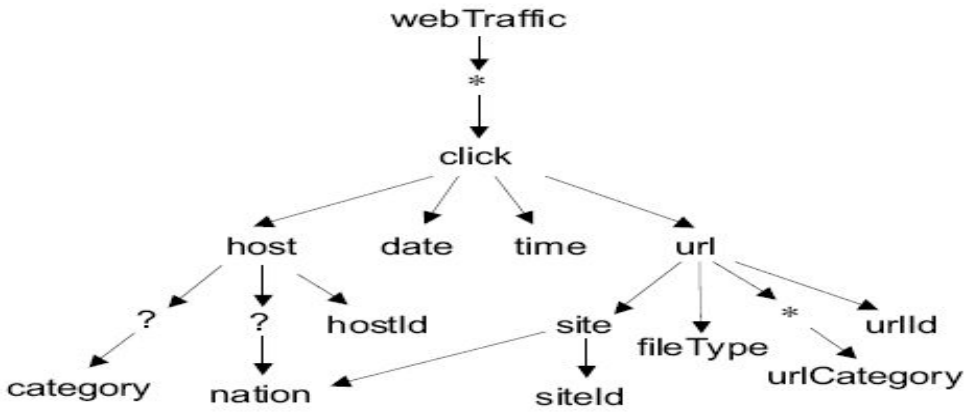


Figure 4.23 : Le graphe DTD [98].

Les sommets du graphe DTD correspondent aux éléments, attributs et opérations d'une DTD, on remarquera que les attributs et les éléments sont représentés de la même manière car la méthode suppose qu'ils sont équivalents.

c. Choisir les faits

Le concepteur va choisir un ou plusieurs sommets qui représenteront les faits, chacun de ces faits représentera la racine du schéma fait/dimension.

d. Construire l'arbre d'attributs

Les sommets de l'arbre d'attributs est un sous ensemble du graphe DTD. L'algorithme qui construit cet arbre est décrit dans la figure 4.24.

L'arbre d'attributs est initialisé avec le sommet fait F, puis cet arbre sera développé par récursivité en utilisant les dépendances fonctionnelles entre les sommets du graphe DTD (En utilisant la procédure expand décrite dans l'algorithme de la figure 4.24).

a- Pour chaque sommet W fils de E dans le graphe DTD :

- Si W est un élément ou un attribut, il sera ajouté comme fils de E dans l'arbre d'attributs.

- Si W est l'opérateur « ? » qui représente la cardinalité 0:1, ses fils seront ajoutés à l'arbre d'attributs comme fils de E, sinon aucun sommet ne sera ajouté.

b- Pour chaque sommet Z qui est le père de E dans le graphe DTD :

- Lorsque on examine les relations dans cette direction, les sommets qui correspondent aux opérateurs « * » et « ? » seront ignorés car ils n'ont aucune signification, de ce fait il faut examiner le contenu des fichiers XML

en utilisant des requêtes XML (En utilisant la procédure checkToMany qui calcule le nombre de valeurs distincts de Z correspondant à chaque valeur de E).

Si la cardinalité détectée est 0:N ou 1:N alors le sommet Z ne sera pas ajouté à l'arbre d'attributs, dans le cas ou on trouve la cardinalité 0:1 ou bien 1:1 alors l'avis du concepteur est nécessaire (En utilisant la procédure askDesignerToOne).

```
root=newVertex(F);
// newVertex(<vertex>) returns a new vertex
// of the attribute tree
// corresponding to <vertex>
// of the DTD graph
expand(F,root);


expand(E,V):
// E is the current DTD vertex,
// V is the current attribute tree vertex
{  for each child W of E do
      if W is element or attribute do
      {  next=newVertex(W);
         addChild(V,next);
         // adds child W to V
         expand(W,next);
      }
      else
         if W="?" do
            expand(W,V);
   for each parent Z of E such that
          Z is not a document element do
      if Z="?" or Z="*" do
         expand(Z,V);
      else
         if not toMany(E,Z) do
            if askDesignerToOne(E,Z) do
            {  next=newVertex(Z);
               addChild(V,next);
               expand(Z,next);
            }
}
```

Figure 4.24 : Un algorithme pour la construction d'un arbre d'attributs [98].

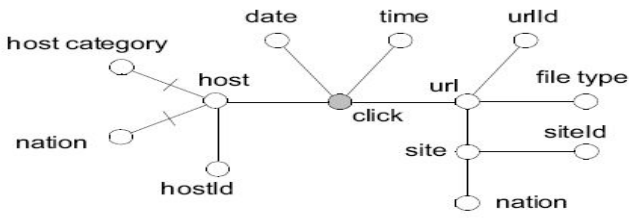


Figure 4.25 : L'arbre d'attributs dérivé du graphe de DTD [98].

4.6. Conclusion

Les auteurs du papier **[95]** présentent un framework utilisant la technologie du web sémantique OWL/RDF qui permet d'intégrer plusieurs sources de données dans une analyse OLAP basée sur une approche de conception dirigée par les besoins d'utilisateurs. Les auteurs utilisent des ontologies OWL/RDF pour uniformiser leurs données hétérogènes. Le farmework proposé utilise un langage d'interrogation déclaratif (RQL) pour construire les cubes OLAP à partir de plusieurs sources de données.

Les auteurs du papier **[96]** exposent une méthodologie de conception de cubes OLAP basée sur les ontologies. Cette approche est dirigée par les besoins utilisateurs et les sources de données. Les auteurs formalisent leur conception en se basant sur le modèle relationnel, la relation universelle, le modèle RDF et le modèle OLAP. En fin, un outil complet a été conçu et testé avec un banc de tests.

Dans le papier **[97]**, les auteurs présentent une méthode semi-automatique pour déterminer les concepts multidimensionnels à partir d'une ontologie en entrée qui représente le domaine d'étude. Cette approche est dirigée par les besoins utilisateurs, elle utilise des algorithmes basés sur des heuristiques pour déterminer le modèle multidimensionnel, et suppose un ensemble de règles basées sur les cardinalités. Cependant, l'approche proposée ne présente pas de banc de tests. De ce fait, la faisabilité de ces algorithmes n'est pas prouvée.

M. Gofarelli, S. Rizzi et B. Vrdoljak **[98]** ont étudié l'utilisation des sources XML dans un environnement d'entrepôts de données. Les auteurs notent l'importance de XML dans le cas où le web devient une partie importante du processus décisionnel des entreprises. Les auteurs proposent une approche dirigée par les sources de données pour automatiser la conception d'un entrepôt de données utilisant des DTD comme point de départ. Ils assoient leur démarche sur un ensemble d'algorithmes basé sur des règles d'extraction des relations « père/fils » dans un fichier DTD. Ce qui permettra de transformer cette DTD en un modèle multidimensionnel.

Dans ce chapitre, nous avons présenté les différents types d'approches de conception à savoir celles guidées par les sources et celles guidées par les besoins. Suite à cette étude, il apparaît que la conception de schéma d'ED est une tâche complexe, elle exige plusieurs efforts et une bonne expertise du concepteur, tant au niveau des modèles des sources de données qu'en compétence en définition des besoins analytiques, afin de construire un schéma exploitable et qui ne provoque pas l'échec du projet décisionnel.
Ainsi, l'enjeu réside dans le fait d'assister le concepteur tout au long de son projet décisionnel et spécialement la phase conceptuelle du fait qu'elle représente l'un des points clés de ce dernier. Une bonne conception d'un SID doit tenir compte de certains aspects essentiels pour la réussite du projet décisionnel, à l'instar de la

spécification des besoins décisionnels, la modélisation multidimensionnelle, l'hétérogénéité des sources de données et les contraintes de qualité.
Le principal objectif de notre travail consiste à capturer toutes les règles qui peuvent aider à une automatisation de la conception d'un entrepôt de données XML et d'en proposer une KR sous forme d'une ontologie pour les concepteurs et pour les logiciels.

CHAPITRE 5
NOTRE APPROCHE DE CONCEPTION AUTOMATIQUE D'UN ENTREPOT DE DONNEES XML

5.1. Introduction

Nous avons présenté ci-dessus tous les ingrédients nécessaires à la compréhension de notre travail. Nous avons évoqué la modélisation multidimensionnelle de données, la technologie XML et le domaine d'ontologie.
Nous avons réalisé une étude approfondie de l'état de l'art des approches d'automatisation de la conception des entrepôts de données. Elle nous a permis de capturer les règles utilisées pour établir l'automatisation. En prenant comme base cette analyse, nous allons proposer une représentation de connaissances sous forme d'une ontologie pour l'aide à la conception automatisée d'un entrepôt de données XML, nous allons ensuite l'implémenter.

On trouve dans la littérature plusieurs approches qui ont pour but l'automatisation de la conception d'un entrepôt de données multidimensionnel.
Cependant, les règles d'automatisation ne sont pas bien identifiées, ni bien définies et ni bien formalisées. C'est dans ce contexte que se positionnent les travaux que nous présentons dans ce chapitre. Il s'agit précisément de chercher toutes ces connaissances de conception pour définir une représentation des connaissances, puis formaliser cette dernière en une ontologie. Pour concevoir cette dernière, nous allons dans un premier temps extraire toutes les connaissances relatives aux approches d'aide à la conception automatisée d'un entrepôt de données **[95] [96] [97] [98]**. Puis nous organiserons ces connaissances dans une structure hiérarchique tout en essayant d'intégrer des ontologies déjà existantes, comme l'ontologie globale OLAP **[95]**. Dans notre implémentation, nous utiliserons le langage OWL **[102] [104]** qui est un langage de représentation d'ontologie sur le web, recommandé par le W3C. Comme ce langage est basé sur les logiques descriptives **[101]**, il ne peut pas y avoir d'ambiguïté dus à l'inconsistance de la terminologie. Il est employé dans notre étude pour décrire les connaissances qui concernent le domaine de l'automatisation d'une conception d'un entrepôt de données XML.
Malgré que le langage OWL soit bien adapté pour représenter la structure des connaissances telle que les classes, les propriétés et les taxonomies, il ne peut pas représenter une connaissance déductive, qui est sous forme d'une règle. De ce fait, le SWRL **[103]**, qui est un langage de règles basé sur OWL, sera utilisé pour exprimer divers contraintes sur les connaissances.

Nous coderons cette ontologie ainsi que les règles SWRL en utilisant l'éditeur Protégé **[88]**, et nous utiliserons Pellet **[92]** comme outil de vérification et SQWRL comme moteur de requêtes. Nous utiliserons également le système JESS **[105]**, comme moteur de règles en transformant l'ontologie exprimée en OWL ainsi que les contraintes SWRL en faits JESS et règles JESS respectivement.

5.2. Notre approche de conception automatique

On suppose que notre approche XEDONTO **[106]** utilisera des schémas XML non normalisés comme point de départ, vu qu'ils permettent une représentation plus expressive qu'une DTD, en particulier les cardinalités avec l'utilisation des contraintes minOccurs et maxOccurs. Cette approche utilise aussi le modèle Dimension/Fait pour représenter le modèle multidimensionnel. On s'intéresse beaucoup aux cardinalités, au type de données ainsi qu'a des fonctions heuristiques pour transformer un schéma XML non normalisé en un modèle multidimensionnel sous forme d'un schéma XML.

La recherche des règles de transformation d'un schéma XML non normalisé directement vers un modèle multidimensionnel est une tâche fastidieuse. Car il est difficile d'avoir une compréhension rapide et globale d'un vocabulaire XML basé sur un schéma XML du fait de la complexité de son formalisme. C'est pour cela qu'il est intéressant de représenter ce vocabulaire en un ensemble de dépendances fonctionnelles (*EnsDF*) **[107]**.

La notion de dépendance fonctionnelle fut introduite par CODD **[107]** afin de caractériser des relations qui peuvent être décomposées sans perte d'information. Plus simplement, un attribut (ou groupe d'attributs) Y dépend fonctionnellement d'un attribut (ou group d'attributs) X, si, étant donné une valeur de X, il lui correspond une valeur unique de Y (et ceci quel que soit l'instant considéré).
Ensuite, cet ensemble EnsDF sera décomposé en troisième forme normale donnant un autre ensemble de dépendances fonctionnelles normalisé (*EnsDFN*) en utilisant l'algorithme de synthèse d'Amstrong **[109]**. Les trois premières formes normales ont pour objectif de permettre une décomposition de relation sans perdre d'informations, à partir de la notion de dépendance fonctionnelle **[108]**. L'objectif de cette décomposition est d'aboutir à un schéma conceptuel représentant les entités et les associations canoniques.
La première forme normale permet simplement d'obtenir des tables rectangulaires, une relation est en première forme normale si tout attribut contient une valeur atomique **[108]**.
La deuxième forme normale permet d'assurer l'élimination de certaines redondances en garantissant qu'aucun attribut n'est déterminé seulement par une partie de la clé, une relation est en deuxième forme normale si et seulement si elle est en première forme et tout attribut n'appartenant pas à une clé ne dépend pas

que d'une partie de cette clé **[108]**.
La troisième forme normale permet d'assurer l'élimination des redondances dues aux dépendances transitives, une relation est en troisième forme normale si et seulement si elle est en deuxième forme et tout attribut n'appartenant pas à une clé ne dépend pas d'un attribut non clé **[108]**.
Cet ensemble EnsDFN sera injecté dans la base de connaissances pour en déduire le modèle multidimensionnel ainsi qu'un nouveau schéma XML basé sur des concepts multidimensionnels. La figure 5.1 présente un schéma global de l'architecture du système d'aide à la conception automatisée d'un entrepôt de données XML.

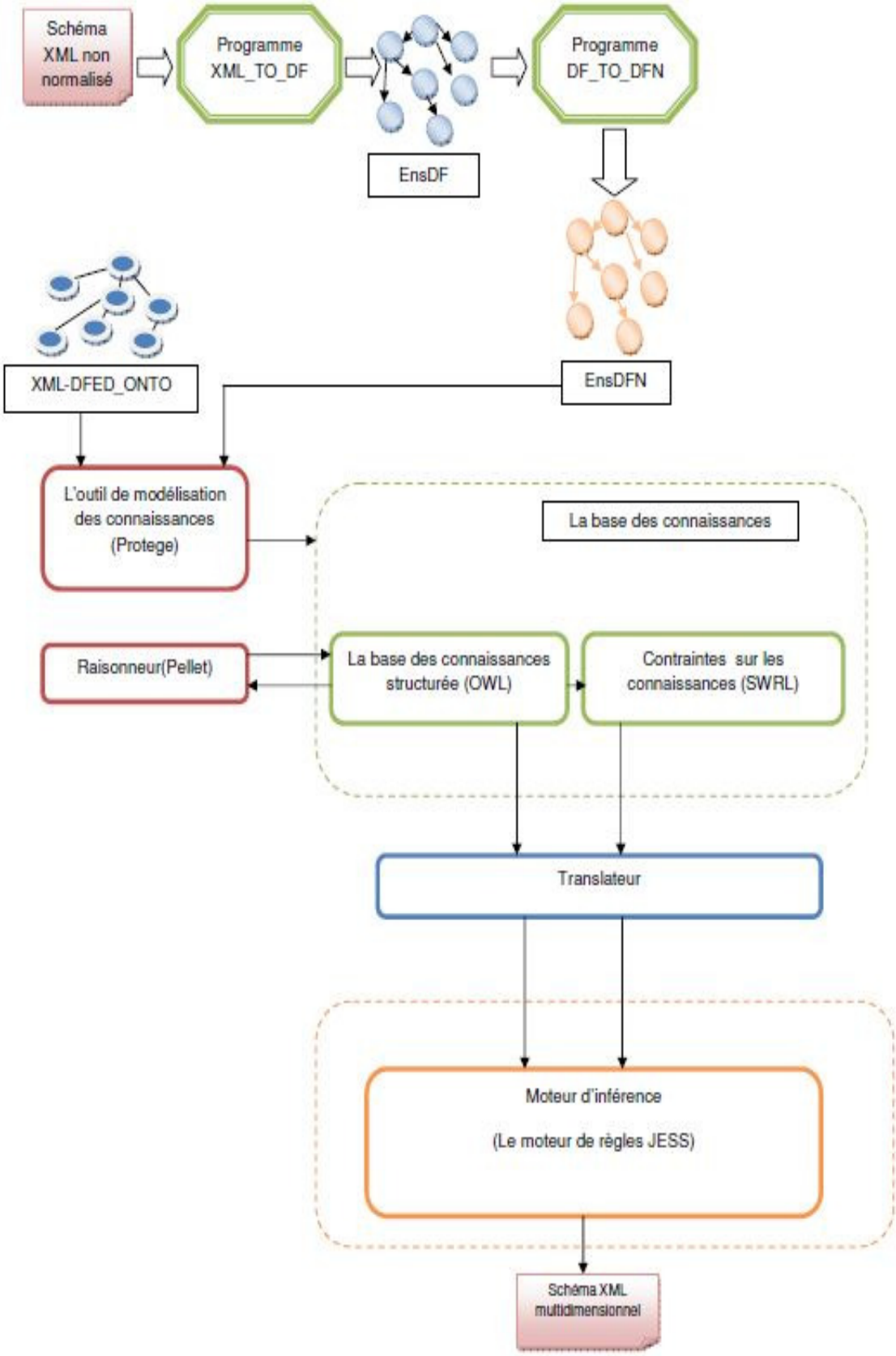


Figure 5.1 : Architecture du système.

Pour illustrer notre approche XEDONTO, nous allons présenter tout au long du chapitre un exemple qui portera sur l'étude de la vente des produits à des clients, on suppose que le schéma XML non normalisé « SchemaXMLVente.xsd » en entrée sera comme suit :

```
<schema version="1.0"
  xmlns="http://www.w3.org/2001/XMLSchema"
  xmlns:ONTOEXP ="http://XMLDF-ED_ONTO.org/Exemples"
  targetNamespace="http://XMLDF-ED_ONTO.org/Exemples"
  elementFormDefault="qualified"
>

  <complexType name="Client" >
    <sequence>
      <element name="codeClient" type="integer" />
      <element name="nomClient" type="string" />
      <element name="prenomClient" type="string" />
      <element name="Localisation">
        <complexType>
          <sequence>
              <element name="ville" type="string" />
              <element name="pays" type="string" />
          </sequence>
        </complexType>
    </sequence>
  </complexType>

  <complexType name="Produit" >
    <sequence>
      <element name="codeProduit" type="integer" />
      <element name="description" type="string" />
      <element name="prixUnitaire" type="integer" />
      <element name="TypeProduit">
        <complexType>
          <sequence>
              <element name="gamme" type="string"  />
              <element name="categ" type="string" />
          </sequence>
        </complexType>
    </sequence>
  </complexType>

  <complexType name="Date" >
    <sequence>
      <element name="codeD" type="integer" />
      <element name="mois" type="integer" />
      <element name="trimestre" type="integer" />
      <element name="annee" type="integer" />
    </sequence>
  </complexType>
```

```
<complexType name="Vente" >
  <sequence>
    <element name="codeClient" type="integer" />
    <element name="codeProduit" type="integer" />
    <element name="codeD" type="integer" />
    <element name="montantV" type="integer" />
    <element name="qteV" type="integer" />
  </sequence>
</complexType>

<complexType name="FichierVente" >
  <sequence>
    <element name="Client" type="ONTOEXP:Client"
      minOccurs="0" maxOccurs="unbounded" />
    <element name="flow" type="ONTOEXP:Produit"
      minOccurs="0" maxOccurs="unbounded" />
    <element name="flow" type="ONTOEXP:Date"
      minOccurs="0" maxOccurs="unbounded" />
    <element name="flow" type="ONTOEXP:Vente"
      minOccurs="0" maxOccurs="unbounded" />
  </sequence>
</complexType>


<element name="fichierVente" type="ONTOEXP:FichierVente" >

  <key name="cleClient" >
    <selector xpath=".//ONTOEXP:Client" />
    <field xpath="ONTOEXP:codeClient" />
  </key>
  <key name="CleLocalisation" >
    <selector xpath=".//ONTOEXP:Client:Localisation" />
    <field xpath="ONTOEXP:ville" />
  </key>
  <key name="CleProduit" >
    <selector xpath=".//ONTOEXP:Produit" />
    <field xpath="ONTOEXP:codeProduit" />
  </key>
  <key name="CleTypeProduit" >
    <selector xpath=".//ONTOEXP:Produit:TypeProduit" />
    <field xpath="ONTOEXP:gamme" />
  </key>
 <key name="CleDate" >
    <selector xpath=".//ONTOEXP:Date" />
    <field xpath="ONTOEXP:codeD" />
 </key>
 <key name="cleVente" >
    <selector xpath=".//ONTOEXP:Vente" />
    <field xpath="ONTOEXP:codeClient" />
```

```
    <field xpath="ONTOEXP:codeProduit" />
    <field xpath="ONTOEXP:codeD" />
  </key>


  <keyref name="codeClientFK" refer="ONTOEXP:cleClient" >
    <selector xpath="ONTOEXP:Vente/Client:codeClient" />
    <field xpath="." />
  </keyref>
   <keyref name="codeProduitFK" refer="ONTOEXP:cleProduit" >
    <selector xpath="ONTOEXP:Vente/ONTOEXP:codeProduit" />
    <field xpath="." />
  </keyref>
     <keyref name="CodeDateFK" refer="ONTOEXP:cleDate" >
    <selector xpath="ONTOEXP:Vente/ONTOEXP:CodeD" />
    <field xpath="." />
  </keyref>

 </element>

</schema>
```

5.2.1. Étape d'extraction et de normalisation

La première phase de cette étape consiste à extraire, du schéma XML non normalisé en entrée, l'ensemble des dépendances fonctionnelles possible en se basant sur des règles identifiées sur la structuration des relations dans un schéma XML.

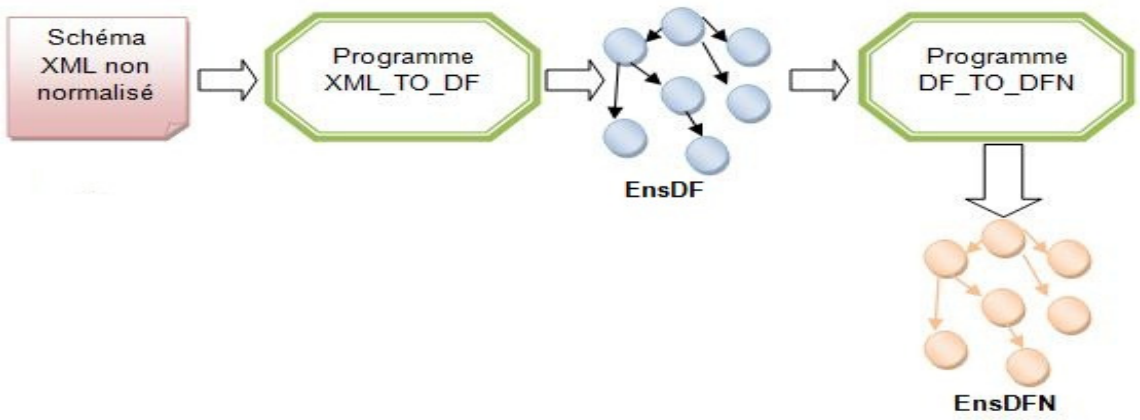


Figure 5.2 : L'étape d'extraction et de normalisation.

Un schéma XML peut présenter deux structurations de relations **[98]** : celle qui utilise des sous éléments et une autre qui manipule les éléments Key et KeyRef.

Nous nous sommes intéressés à cette dernière puisse qu'elle permet d'avoir une structuration plus complexe, et de reproduire les concepts de clé primaire et de clé étrangère du domaine des bases de données.
Le mot-clé Key permet d'indiquer qu'un attribut ou un élément est unique non nul. Tandis que KeyRef offre la possibilité de référencier d'autres clés. Dans le tableau ci-dessous, nous donnons quelques règles permettant le mappage de la structure d'un schéma XML vers l'ensemble de DFs équivalent EnsDF.

Schéma XML	**Dépendance fonctionnelle**	**Schéma relationnel**
R1 : Un élément de type complexe Cx ayant une clé K et un ensemble d'éléments simples E1….En sachant que l'ensemble de ses éléments à comme cardinalité minOccurs = 0/1 et maxOccurs = 1.	K→E1, E2, …. En	-Une table rectangulaire Cx ayant les attributs K, E1, …En -K est une clé primaire de la table Cx
R2 : Un élément de type complexe C1x composé d'un élément de type complexe C2x sachant que la cardinalité de K1 est minOccurs = 1 et maxOccurs = 1 et la cardinalité de K2 est minOccurs = 0/1 et maxOccurs = 1, K1 et K2 les deux clés des deux éléments C1x et C2x respectivement.	K1→K2	-K2 est une clé étrangère qui fait référence à la table C2x

Tableau 5.1 : Les règles de transformation d'un schéma XML vers EnsDF.

Nous avons implémenté notre démarche. Nous avons produit un premier programme, appelé «XML_TO_DF», qui utilise l'API SAX basé sur un modèle événementiel pour analyser le schéma XML en entrée.
Voyons comment appliquer ces règles sur le schéma XML « SchemaXMLVente.xsd » pour extraire l'ensemble EnsDF, ci-dessous un tableau qui décrit le processus d'extraction de l'ensemble des dépendances fonctionnelles :

Script XML	EnsDF	Règle
<complexType name="Client" > <sequence> <element name="codeClient" type="integer" /> <element name="nomClient" type="string" /> <element name="prenomClient" type="string" /> <element name="Localisation"> <complexType> <sequence> <element name="ville" type="string" /> <element name="pays" type="string" /> </sequence> </complexType> </sequence> </complexType> <key name="cleClient" > <selector xpath=".//ONTOEXP:Client" /> <field xpath="ONTOEXP:codeClient" /> </key> <key name="CleLocalisation" > <selector xpath=".//ONTOEXP:Client:Localisation" /> <field xpath="ONTOEXP:ville" /> </key>	codeClient→nomClient codeClient→prenomClient codeClient→pays codeClient→ville ville→pays	R1 R1 R1 R2 R1
<complexType name="Produit" > <sequence> <element name="codeProduit" type="integer" /> <element name="description" type="string" /> <element name="prixUnitaire" type="integer" /> <element name="TypeProduit"> <complexType> <sequence> <element name="gamme" type="string" /> <element name="categ" type="string" /> </sequence> </complexType> </sequence> </complexType> <key name="CleProduit" > <selector xpath=".//ONTOEXP:Produit" /> <field xpath="ONTOEXP:codeProduit" /> </key> <key name="CleTypeProduit" > <selector xpath=".//ONTOEXP:Produit:TypeProduit" /> <field xpath="ONTOEXP:gamme" /> </key>	codeProduit→description codeProduit→prixUnitaire codeProduit→categ codeProduit →gamme gamme→categ	R1 R1 R1 R2 R1

<complexType name="Date" > <sequence> <element name="codeD" type="integer" /> <element name="mois" type="integer" /> <element name="trimestre" type="integer" /> <element name="annee" type="integer" /> </sequence> </complexType> <key name="CleDate" > <selector xpath=".//ONTOEXP:Date" /> <field xpath="ONTOEXP:codeD" /> </key>	codeD→mois codeD→trimestre codeD→annee	R1 R1 R1
<complexType name="Vente" > <sequence> <element name="codeClient" type="integer" /> <element name="codeProduit" type="integer" /> <element name="codeD" type="integer" /> <element name="montantV" type="integer" /> <element name="qteV" type="integer" /> </sequence> </complexType> <key name="cleVente" > <selector xpath=".//ONTOEXP:Vente" /> <field xpath="ONTOEXP:codeClient" /> <field xpath="ONTOEXP:codeProduit" /> <field xpath="ONTOEXP:codeD" /> </key> <keyref name="codeClientFK" refer="ONTOEXP:cleClient" > <selector xpath="ONTOEXP:Vente/Client:codeClient" /> <field xpath="." /> </keyref> <keyref name="codeProduitFK" refer="ONTOEXP:cleProduit" > <selector xpath="ONTOEXP:Vente/ONTOEXP:codeProduit" /> <field xpath="." /> </keyref> <keyref name="CodeDateFK" refer="ONTOEXP:cleDate" > <selector xpath="ONTOEXP:Vente/ONTOEXP:CodeD" /> <field xpath="." /> </keyref>	codeClient,CodeProduit, CodeD→montantV codeClient,CodeProduit, CodeD→qteV	R1 R1

Tableau 5.2 : Extraction de l'ensemble EnsDF du schéma XML «SchemaXMLVente.xsd».

Un deuxième programme consiste à transformer l'ensemble EnsDF en un ensemble de dépendances fonctionnelles normalisées EnsDFN. Ce programme est basé sur l'approche par synthèse d'Amstrong **[109]** pour la normalisation d'un schéma relationnelle décrit par un ensemble de dépendances fonctionnelles.
Nous présentons ci-dessous, l'algorithme de normalisation par synthèse **[109]**, puis nous montrons son déroulement sur l'ensemble EnsDF extrait à partir du schéma XML « SchemaXMLVente.xsd ».

Entrée : L'ensemble A des attributs des DFs de EnsDF. L'ensemble EnsDF.
Sortie : EnsDFN.

Début :
E1 : Suppression des attributs redondants.
E2 : Suppression des DFs redondantes.
E3 : Regroupement des DFs ayant la même partie gauche.
E4: Regroupement des ensembles E_i et E_k dans le cas où on a les dépendances X→Y et Y→X avec X→Y appartient à E_i et Y→X appartient à E_k.
Fin.

Ci-dessous, le déroulement de l'algorithme de normalisation par synthèse **[109]** sur l'ensemble EnsDF du schéma XML « SchemaXMLVente.xsd » :

EnsDF = {
df1:codeClient→nomClient, **df2**:codeClient→prenomClient,
df3:codeClient→pays, **df4**:codeClient→ville, **df5**:ville→pays,
df6:codeProduit→description, **df7**:codeProduit→prixUnitaire,
df8:codeProduit→categ, **df9**:codeProduit →gamme,
df10:gamme→categ,
df11:codeD→mois, **df12**:codeD→trimestre, **df13**:codeD→annee,
df14:codeClient,CodeProduit, CodeD→montantV,
df15:codeClient,CodeProduit, CodeD→qteV
}

Etape de l'algorithme	Résultat
E1	Aucun changement vu qu'il n'y a pas d'attributs redondants.
E2	Suppression des dépendances fonctionnelles df3 et df8 car elles sont transitives.
E3	Création des ensembles suivants : E_1 : {df1, df2, df4} E_2 : {df5} E_3 : {df6, df7, df9} E_4 : {df10} E_5 : {df11, df12, df13} E_6 : {df14, df15}
E4	Aucun changement

Tableau 5.3 : Déroulement de l'algorithme de normalisation par synthèse sur l'ensemble EnsDF du schéma XML « SchemaXMLVente.xsd ».

L'ensemble EnsDFN sera comme suit :
EnsDFN = {
df1' : codeClient→nomClient, prenomClient, ville ;
df2' : ville→pays ;
df3' : codeProduit→description, prixUnitaire, gamme ;
df4' : gamme→categ ;
df5' : codeD→mois, trimestre, annee ;
df6' : codeClient,CodeProduit, CodeD→montantV, qteV
}

5.2.2. Construction du modèle multidimensionnel

Nous présentons dans cette section, la conception et l'implémentation de la représentation des connaissances formalisée par une ontologie ainsi que le processus de son exploitation.

5.2.2.1. Formalisation de l'ontologie « XML-DFED ONTO »

L'objectif principal de la conception d'une ontologie pour l'aide à la conception automatisée d'un entrepôt de données XML est d'avoir une base de connaissances commune partagée par la communauté des concepteurs des entrepôts de données sous la forme d'une ontologie. Cette dernière regroupera les concepts d'un schéma XML, d'une dépendance fonctionnelle et d'un modèle multidimensionnels ainsi que les relations entre ces concepts. Cette ontologie sera utilisée par les experts d'entrepôts de données ainsi que par les logiciels

spécialisés dans la conception des entrepôts de données. L'ontologie, que nous nommons «XML-DFED_ONTO» est formellement conçue sur l'étude des approches d'aide à l'automatisation de la conception des entrepôts de données **[95] [96] [97] [98]**. Cette ontologie a pour but d'obtenir les concepts multidimensionnels par inférence, sachant que celle-ci est alimentée par l'ensemble EnsDFN sous un format précis. L'ontologie « XML-DFED_ONTO » est construite à l'aide de Protégé **[88]**. C'est un outil pour la création, la visualisation et la manipulation des ontologies dans différentes formats de représentations. En utilisant OntoViz **[110]**, un plugin pour Protégé, l'ontologie « XML-DFED_ONTO » est visualisée dans la figure 5.5.

L'ontologie « XML-DFED_ONTO » identifie les concepts existants dans un schéma XML, une dépendance fonctionnelle et un modèle multidimensionnel ainsi que les relations entre l'ensemble de ces concepts. Nous citons, ci-dessous, quelques connaissances **[51] [95] [96] [97] [98] [108]** qui donneront naissance à des concepts et à des relations dans l'ontologie décrite dans la figure 5.4, où les carrés représentent les classes et les flèches les relations.

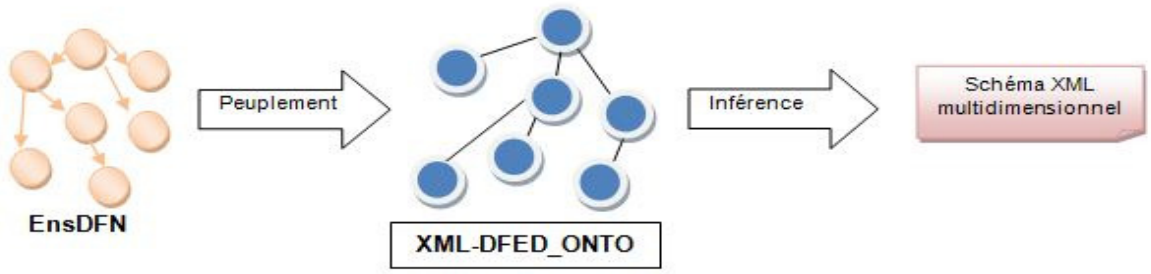


Figure 5.3 : Construction du modèle multidimensionnel.

C1 Une relation en étoile (StarRelation) est composée (hasFD) d'une et une seule dépendance fonctionnelle (FD) normalisée (Connaissances C2.2 et C2.4) **[96]**.

C2 Une dépendance fonctionnelle (FD) est composée (hasBody) d'une et une seule prémisse (Body) et (hasHead) d'un et un seul résultat (Head) **[108]**.

C3 Le résultat (Head) d'une dépendance fonctionnelle dépend de la prémisse (Body) de cette dernière **[108]**.

C4 La prémisse (Body) d'une dépendance fonctionnelle (FD) est une clé (Key) dans un élément XML complexe (XS_Complex_Element).

C5 Le résultat (Head) et la prémisse (Body) d'une dépendance fonctionnelle sont composés d'un ensemble de champs (Field) **[108]**.

C6 Un cube OLAP (OLAPCube) est une relation en étoile (StarRelation) (Connaissance C2.4) **[96]**.

C7 Un cube OLAP (OLAPCube) est composé (hasFact) d'un fait OLAP (OLAPFact) (Connaissance C1.1) **[95]**.

C8 Un fait OLAP est une dépendance fonctionnelle (Connaissances C2.2, C2.3) **[96]**.
C9 Un fait OLAP (OLAPFact) est composé d'un ensemble de dimensions (OLAPDimension) et d'un ensemble de mesure (OLAPMeasure) (Connaissance C1.2) **[95]**.
C10 La prémisse (Body) d'une dépendance fonctionnelle forme l'ensemble des dimensions d'un fait OLAP (Connaissance C2.2) **[96]**.
C11 Le résultat (Head) d'une dépendance fonctionnelle forme l'ensemble des mesures d'un fait OLAP (Connaissance C2.2) **[96]**.
C12 Un schéma XML multidimensionnel (XS_Schema) est structuré en un ensemble d'éléments complexes (XS_Complex_Element) **[51]**.
C13 Un élément complexe (XS_Complex_Element) est composé (hasElt) d'éléments simples (XS_Element) **[51]**.
C14 Chaque élément complexe comprend une clé (Key) représentée par une prémisse (Body) d'une dépendance fonctionnelle (FD).
C15 Un élément complexe est une dépendance fonctionnelle.
C16 Un élément (XS_Element) est un champ(Field).

Afin d'exploiter cette ontologie, nous supposons que :

- L'ontologie XML–DFED_ONTO sera alimentée par des instances qui regroupent l'ensemble ENSDFN généré par l'étape d'extraction et de normalisation.

- Une inférence sur cette ontologie sera faite pour générer l'ensemble des concepts multidimensionnels ainsi que les concepts XML qui formeront un schéma XML multidimensionnel.

5.2.2.2. Codage de l'ontologie « XML-DFED_ONTO »

a) Codage des connaissances structurelles en OWL

Pour coder notre ontologie, nous utiliserons le langage web d'ontologie OWL **[102] [104]** qui est un langage de représentation d'ontologie sur le web. Il est recommandé par le W3C[1]. Comme ce langage est basé sur les logiques descriptives **[101]**, il ne peut pas y'avoir d'ambiguïté dus à l'inconsistance de la terminologie. Le langage OWL est conçu pour être utilisé par des applications qui ont besoin d'analyser le contenu des informations mais pas seulement de les présenter à l'utilisateur.

[1] http://www.W3C.org

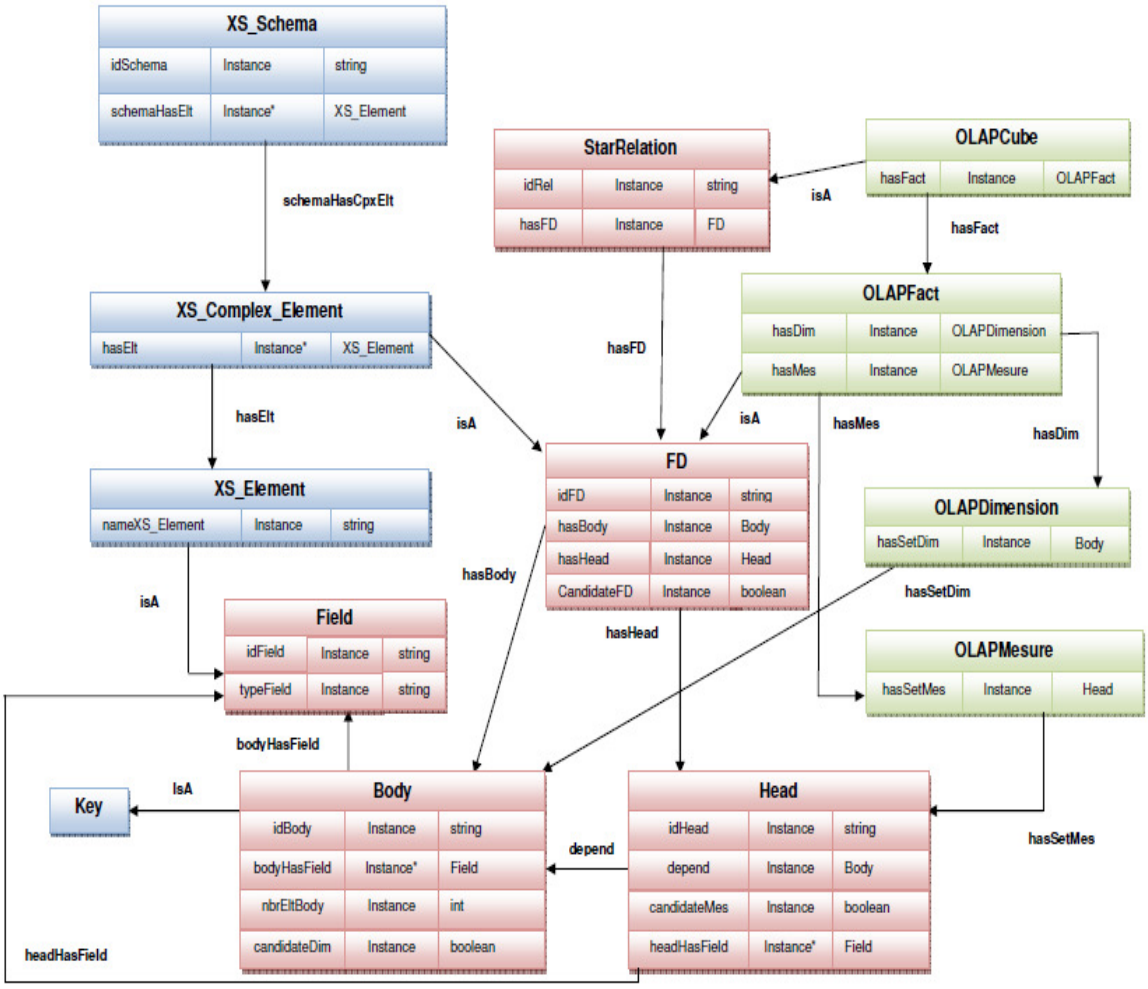


Figure 5.4 : L'ontologie XML-DFED_ONTO.

Récemment le W3C a mis une recommandation pour OWL. Ce langage est utilisé pour représenter les ontologies. Dans les technologies du W3C, on distingue aussi le langage XML qui offre une syntaxe pour les documents structurés mais sans aucune contrainte sémantique. Il y a aussi le RDF[2] qui est un modèle de données pour les ressources et les relations entre ces dernières. Ce modèle fournit une sémantique représentée en XML. Le schéma RDF est un vocabulaire pour la description des propriétés et des classes.

[2] http://www.RDF.org/

Le langage OWL ajoute plus de vocabulaire pour décrire les propriétés et les classes. Il ne se limite pas aux relations entre les classes. Il décrit également les cardinalités, les équités, un typage riche des propriétés, les caractéristiques des propriétés et les classes énumératives. On distingue trois sous langages OWL : OWL-Lite, OWL-DL et OWL-Full. Chacun diffère de l'autre par son expressivité et sa décidabilité. OWL-Full est le plus expressif mais il est indécidable. Nous avons utilisé OWL-DL **[104]** dans notre formalisation car il est décidable et plus expressif que OWL-Lite. Dans la figure 5.5, nous présentons le codage en OWL de la classe qui représente une dépendance fonctionnelle (FD). Nous remarquons que la classe FD est une sous classe de la classe générale «Thing ». Elle est composée de deux propriétés de donnée (idFD) de type « string » et la propriété (candidateFD) de type « boolean » ainsi que deux propriétés de type objet (hasBody) et (hasHead). Cette déclaration est enrichie avec une restriction sur les cardinalités des propriétés (hasBody) et (hasHead). Cette restriction suppose qu'une dépendance fonctionnelle à une et une seule prémisse (Body) et un et un seul résultat (Head). (c.f. la connaissance C2).

```
<owl:Class rdf:ID="FD">
    <rdfs:subClassOf>
      <owl:Restriction>
        <owl:cardinality rdf:datatype="http://www.w3.org/2001/XMLSchema#int"
        >1</owl:cardinality>
        <owl:onProperty>
          <owl:ObjectProperty rdf:ID="hasBody"/>
        </owl:onProperty>
      </owl:Restriction>
    </rdfs:subClassOf>
    <rdfs:subClassOf>
      <owl:Restriction>
        <owl:cardinality rdf:datatype="http://www.w3.org/2001/XMLSchema#int"
        >1</owl:cardinality>
        <owl:onProperty>
          <owl:ObjectProperty rdf:ID="hasHead"/>
        </owl:onProperty>
      </owl:Restriction>
    </rdfs:subClassOf>
    <rdfs:subClassOf rdf:resource="http://www.w3.org/2002/07/owl#Thing"/>
  </owl:Class>

<owl:ObjectProperty rdf:about="#hasBody">
    <rdfs:domain rdf:resource="#FD"/>
    <rdfs:range rdf:resource="#Body"/>
</owl:ObjectProperty>

<owl:ObjectProperty rdf:about="#hasHead">
    <rdfs:range rdf:resource="#Head"/>
    <rdfs:domain rdf:resource="#FD"/>
</owl:ObjectProperty>

<owl:DatatypeProperty rdf:ID="idFD">
    <rdfs:domain rdf:resource="#FD"/>
    <rdfs:range rdf:resource="http://www.w3.org/2001/XMLSchema#string"/>
</owl:DatatypeProperty>

<owl:DatatypeProperty rdf:ID="candidateFD">
    <rdfs:domain rdf:resource="#FD"/>
    <rdfs:range>
      <owl:DataRange>
        <owl:oneOf rdf:parseType="Resource">
          <rdf:rest rdf:parseType="Resource">
            <rdf:first rdf:datatype="http://www.w3.org/2001/XMLSchema#boolean"
            >false</rdf:first>
            <rdf:rest rdf:resource="http://www.w3.org/1999/02/22-rdf-syntax-ns#nil"/>
          </rdf:rest>
          <rdf:first rdf:datatype="http://www.w3.org/2001/XMLSchema#boolean"
          >true</rdf:first>
        </owl:oneOf>
      </owl:DataRange>
    </rdfs:range>
</owl:DatatypeProperty>
```

Figure 5.5 : Le codage en OWL de la classe FD.

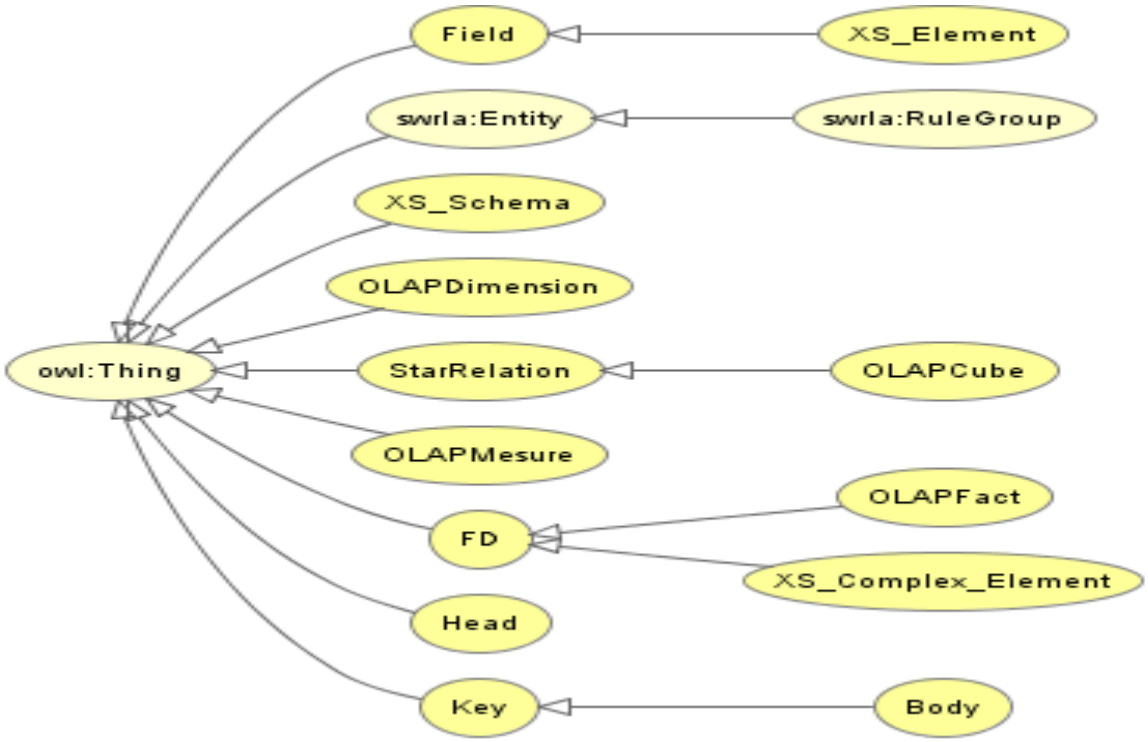


Figure 5.6 : Les classes de l'ontologie XML-DFED_ONTO avec l'outil OWLViz (Protégé).

Si une ontologie est consistante cela veut dire qu'il y' a absence de contradictions dans sa structuration. La consistance de l'ontologie XML-DFED_ONTO a été vérifiée avec le moteur d'inférence Pellet[3] (Figure 5.7). Pellet est un moteur d'inférence pour le langage OWL DL, il est basé sur les algorithmes des tableaux développés pour la logique descriptive.

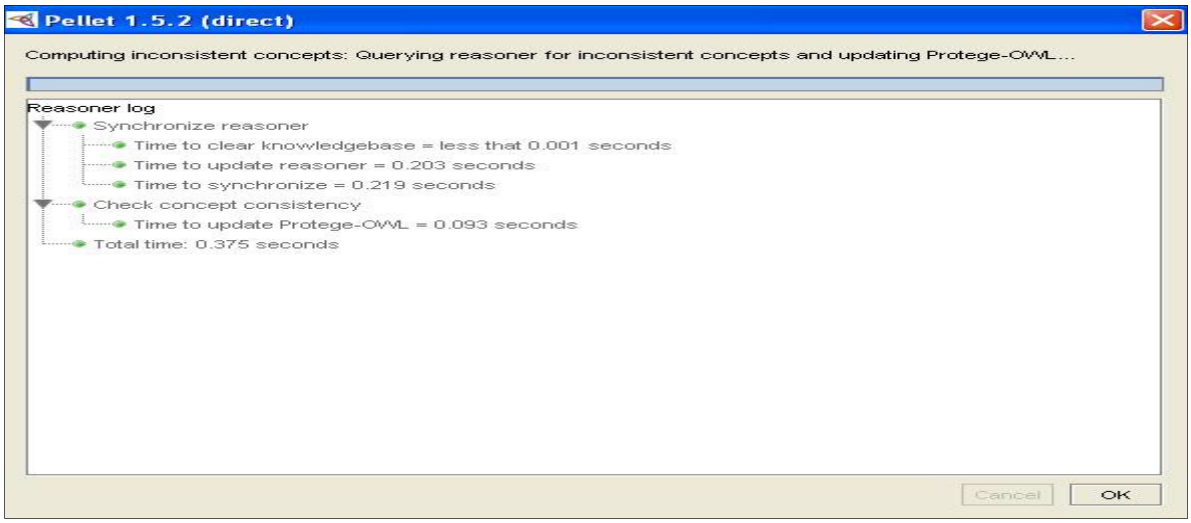


Figure 5.7 : Vérification de la consistance de l'ontologie XML-DFED_ONTO avec le moteur d'inférence Pellet (Protégé).

[3] http://clarkparsia.com/pellet

b) Les règles SWRL

Le SWRL est basé sur une combinaison des langages OWL-DL et OWL-Lite avec les sous langages unaire et binaire Datalog et RuleML **[111]**. Il étend l'ensemble des axiomes OWL pour comprendre les règles Horn-like **[103]**. Ce langage utilise les règles de Horn-like qui ont la forme d'une implication entre un antécédent (Body) et une conséquence (Head). Les conditions spécifiées dans la conséquence seront satisfaites une fois que les conditions de l'antécédent le sont. L'antécédent et la conséquence sont composés de zéro ou de plusieurs atomes. Un antécédent vide est évalué à vrai, lorsqu'il est satisfait par n'importe quelle interprétation. D'où la conséquence est évaluée à vrai aussi par n'importe quelle interprétation. D'autre part, une conséquence vide est évaluée à faux, lorsqu'elle n'est satisfaite par aucune interprétation. Donc, l'antécédent est aussi non satisfait par aucune interprétation. Plusieurs atomes sont traités comme une conjonction **[103]**. A noter qu'une règle avec une conjonction de conséquences peut être facilement transformée en un ensemble de règles, chacune composée d'une conséquence atomique, en utilisant les transformations de Lloyd-Topor **[87]**. Les atomes de ces règles peuvent avoir la forme C(x), P(x,y), sameAs(x,y) ou differentFrom(x,y), où C est une description OWL, P est une propriété OWL et x, y sont des variables, des individus OWL ou bien des valeurs de données OWL **[103]**. La syntaxe d'une règle SWRL est : antécédent→conséquence, où l'antécédent et la conséquence sont des conjonctions d'atomes. Les variables sont indiquées en utilisant une convention standard en les préfixant par un point d'interrogation (e.g., ?x).

Nous présentons dans le tableau ci-dessous les règles SWRL qui permettent d'en déduire l'ensemble des concepts multidimensionnels à partir de l'ensemble des dépendances fonctionnelles normalisées EnsDFN injectées dans la base des connaissances, ces règles doivent être exécutées dans l'ordre cité dans le tableau.

Règle SWRL	Description	Objectif
R1_searchMesCandidate	Head(?h) ∧ Field(?f) ∧ headHasField(?h, ?f) ∧ typeField(?f, ?typeF) ∧ swrlb:contains(?typeF, "int") → candidateMes(?h, true)	Cette règle permet de repérer les éventuelles mesures en se basant sur le type de donnée, elle suppose qu'une mesure doit être un champ (Field) ayant un type entier, la règle met le résultat (Head) contenant ce champ (Field) comme résultat (Head) candidat.
R2_searchDimCandidate	Body(?b) ∧ nbrEltBody(?b, ?NbrEltBody) ∧ swrlb:greaterThanOrEqual(?NbrEltBody, 3) → candidateDim(?b, true)	Cette règle cherche les dimensions candidates, elle suppose que chaque prémisse (Body) d'une dépendance fonctionnelle (FD) contenant trois champs (Field) au minimum sera candidate pour être une dimension.
R3_searchFDCandidate	FD(?fd) ∧ Body(?b) ∧ Head(?h) ∧ depend(?h, ?b) ∧ hasBody(?fd, ?b) ∧ hasHead(?fd, ?h) ∧ candidateDim(?b, true) ∧ candidateMes(?h, true) → candidateFD(?fd, true)	La règle suivante détermine les dépendances fonctionnelles (FD) ayant une prémisse (Body) ainsi qu'un résultat (Head) candidates pour former des cubes OLAP, chaque dépendance fonctionnelle formera un cube OLAP (OLAPCube).
R4_InferOLAPConcepts	FD(?fd) ∧ Body(?b) ∧ Head(?h) ∧ depend(?h, ?b) ∧ hasBody(?fd, ?b) ∧ hasHead(?fd, ?h) ∧ candidateDim(?b, true) ∧ candidateMes(?h, true) ∧ candidateFD(?fd, true) OLAPCube(?oc) ∧ OLAPFact(?of) ∧ OLAPDimension(?od) ∧ OLAPMesure(?om) → hasSetDim(?od, ?b) ∧ hasSetMes(?om, ?h) ∧ hasDim(?of, ?od) ∧ hasMes(?of, ?om) ∧ hasFact(?oc, ?of)	Cette règle permet de déduire l'ensemble des concepts multidimensionnels.
R5_ConstructStarRelation	StarRelation(?sr) ∧ FD(?fd) ∧ candidateFD(?fd, true) → hasFD(?sr, ?fd)	Cette règle permet de déduire les relations en étoiles (StarRelation), chaque relation en étoile est représentée par une dépendance fonctionnelle (FD) candidate.

Tableau 5.4 : Les règles SWRL.

Toutes les contraintes SWRL ont été modélisées avec l'éditeur de règles SWRL SWRLTab **[112]** qui est un plugin de l'outil Protégé (Figure 5.8).

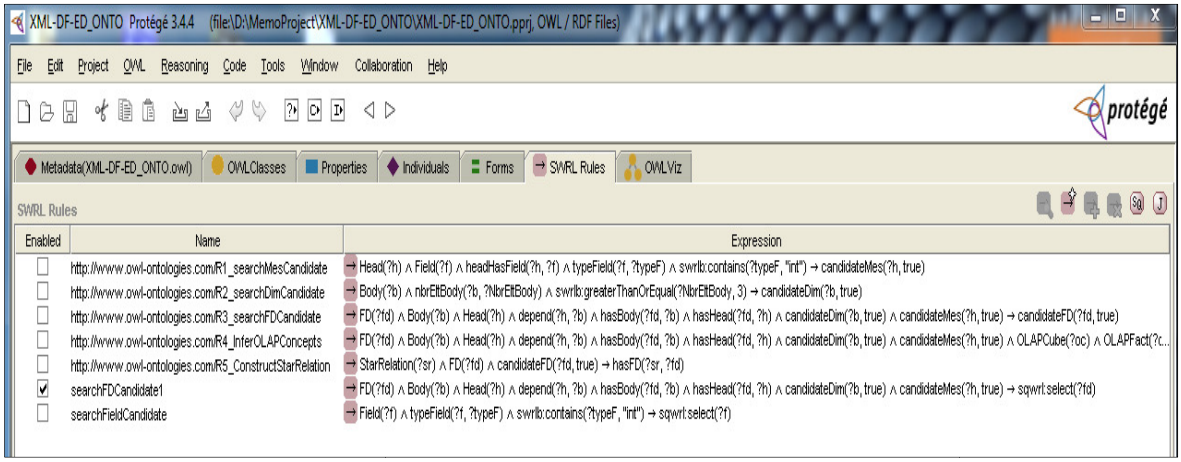


Figure 5.8 : L'écriture des règles SWRL avec le plugin SWRLTab (Protégé).

En résumé, La fonction heuristique formalisée en un ensemble de règles SWRL permet de déduire les concepts multidimensionnels, elle se base sur la recherche des dépendances fonctionnelles satisfaisants les deux conditions suivantes : la première condition exige que le résultat (Head) de cette dépendance fonctionnelle doit contenir au minimum un élément ayant un type entier ce qui représentera la mesure, la deuxième condition est que la prémisse (Body) doit être composé d'au moins trois champs (Field), ces éléments représenterons les dimensions du cube OLAP.

5.2.2.3. Inférence avec le moteur JESS

Dans notre travail, la déduction des concepts multidimensionnels XML est faite en utilisant un moteur à règles nommé JESS[4]. JESS est basé sur le raisonnement à chainage avant. Il implémente l'algorithme Rete **[113]**. JESS contient une base de règles, une mémoire de travail et un moteur d'inférence. Les déductions sont effectuées en comparant les faits en mémoire de travail avec les règles dans la base de règles après avoir transformer les connaissances structurelles OWL en faits JESS ainsi que les contraintes SWRL en règles JESS. Protégé dispose du plugin SWRLTab qui permet un mappage automatique des connaissances OWL et les contraintes SWRL en faits JESS et règles JESS respectivement.

On aura dans le cas de notre exemple, la génération de nouvelle instance dans l'ontologie XML-DFED_ONTO suite au déclenchement de l'ensemble des

[4] http://www.jessrules.com/

règles SWRL, ces instances formeront de nouvelle connaissance construisant le modèle multidimensionnelle, la fonction heuristique proposera la dépendance fonctionnelle **df6**:codeClient, CodeProduit, CodeD→montantV, qteV comme dépendance fonctionnelle candidate à devenir un modèle multidimensionnel vu qu'elle contient un nombre de trois éléments dans la prémisse(Body) (Condition 1 de la fonction heuristique satisfaite) et cette dépendance fonctionnelle contient au minimum un élément de type entier dans la partie droite (Head) (Condition 2 de la fonction heuristique satisfaite), ce qui donne les concepts suivants :
codeClient, CodeProduit et CodeD représenterons les dimensions du cube OLAP.
montantV et qteV formerons l'ensemble des mesures.

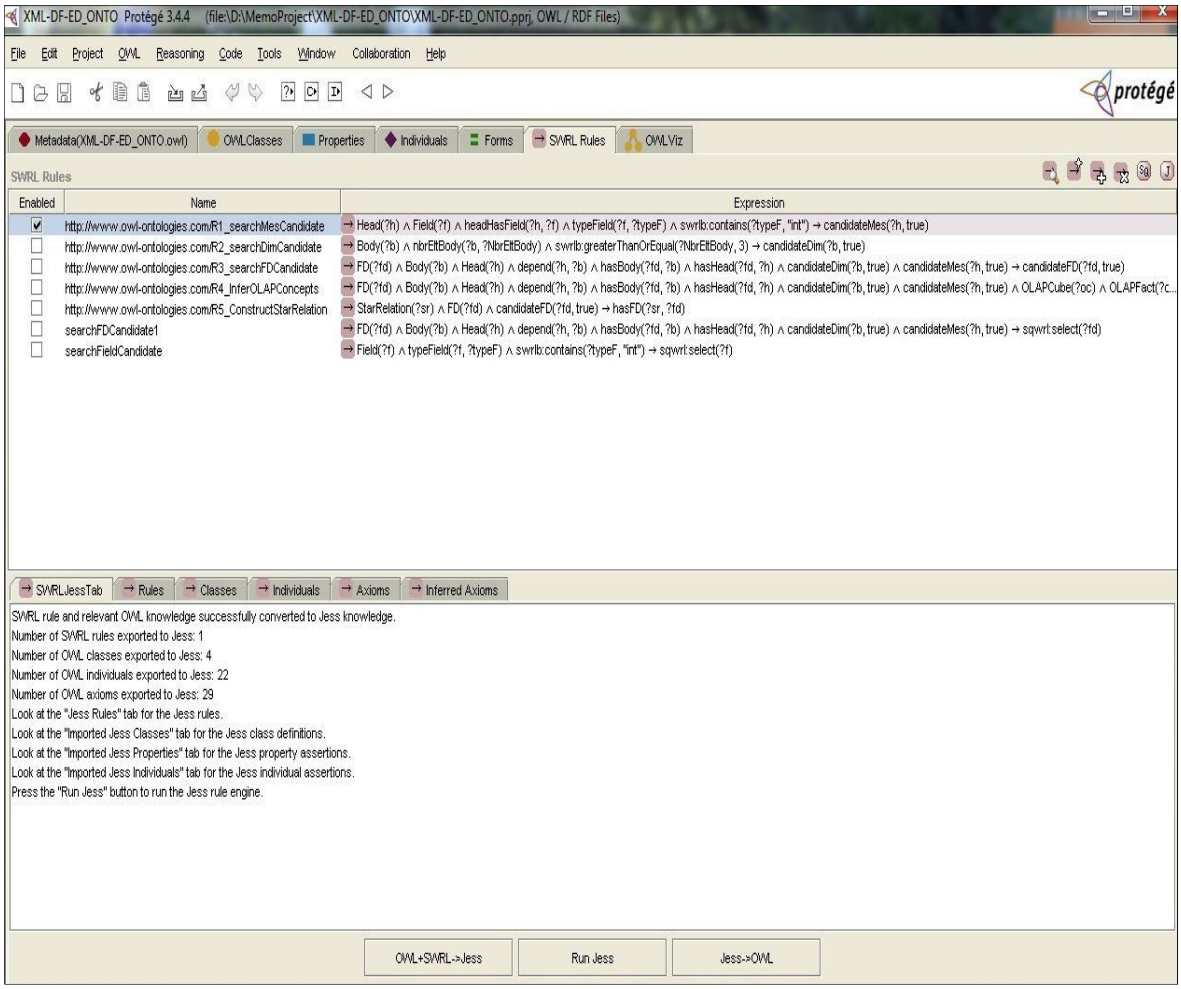


Figure 5.9 : La génération et l'exécution des règles JESS correspondantes à une règle SWRL (Protégé).

Dans la figure 5.10, on présente le schéma XML multidimentionnel résultant. D'après notre ontologie XML-DFED_ONTO on a les deux connnaissances suivantes qui servent à déduire ce schéma XML :

- Un élément (XS_Element) est un champ(Field) (C16).
- Un élément complexe est une dépendance fonctionnelle(C15).

Après inférence avec le moteur JESS, on aurra DF6 comme dépendance focntionnelle condidate pour etre un cube OLAP. Ce qui générera un élément complexe DF6 ayant les éléments codeClient, codeProduit, codeD comme clés et ils representent aussi les dimensions du cube OLAP DF6. Ainsi que les champs montantV et qteV représenteront les mesures qui seront des éléments XML simples.

```
<schema version="1.0"
 xmlns="http://www.w3.org/2001/XMLSchema"
 xmlns:ONTOEXP ="http://XMLDF-ED_ONTO.org/Exemples"
 targetNamespace="http://XMLDF-ED_ONTO.org/Exemples"
 elementFormDefault="qualified"
>

 <complexType name="DF6" >
   <sequence>
     <element name="codeClient_Dim" type="integer" />
     <element name="codeProduit_Dim" type="integer" />
     <element name="codeD_Dim" type="integer" />
     <element name="montantV_Mes" type="integer" />
     <element name="qteV_Mes" type="integer" />
   </sequence>
 </complexType>

 <complexType name="FichierCubeOLAP" >
   <sequence>
      <element name="flow" type="ONTOEXP:DF6"
       minOccurs="0" maxOccurs="unbounded" />
   </sequence>
 </complexType>


 <element name="fichierCubeOLAP" type="ONTOEXP:FichierCubeOLAP" >


  <key name="cleDF6" >
     <selector xpath=".//ONTOEXP:DF6" />
     <field xpath="ONTOEXP:codeClient_Dim" />
     <field xpath="ONTOEXP:codeProduit_Dim" />
     <field xpath="ONTOEXP:codeD_Dim" />
  </key>

 </element>

</schema>
```

Figure 5.10 : Le schéma XML multidimentionnel résultant.

5.3. Implementation

5.3.1. Introduction

Dans cette section, on va présenter en détail l'exécution d'un exemple de schéma XML qui est le schéma « SchemaXMLVente.xsd » décrit dans la section 5.2 avec notre approche de conceptio, en introduisant l'ensemble EnsDFN comme instances dans l'ontologie « XML-DFED_ONTO », le test se ferra avec l'outil Protégé **[88]**. Ce test permettra de vérfier l'aspect fonctionnel de notre approche XEDONTO.

5.3.2. Description du test

Après l'extraction de l'ensemble des dépendances fonctionnelles normalisées EnsDFN du schéma XML « SchemaXMLVente.xsd », on injecte ces dépendances fonctionnelles sous forme d'instances dans l'ontologie en utilisant Protégé.

L'ensemble EnsDFN sera comme suit :

EnsDFN = {

df1' : codeClient→nomClient, prenomClient, ville ;

df2' : ville→pays ;

df3' : codeProduit→description, prixUnitaire, gamme ;

df4' : gamme→categ ;

df5' : codeD→mois, trimestre, annee ;

df6' : codeClient, CodeProduit, CodeD→montantV, qteV

}

Ce qui donne naissance aux instances décrites dans la figure 5.11:

idFD	hasBody					hasHead					candidateFD
	idBody	bodyHasField		nbrEltBody	candidateDim	idHead	headHasField		depend	candidateMes	
		idField	typeField				idField	typeField			
FD1	B1	codeClient	int	1	false	H1	nomClient	str	B1	false	false
							prenomClient	str			
							ville	str			
FD2	B2	ville	str	1	false	H2	pays	str	B2	false	false
FD3	B3	codeProduit	int	1	false	H3	description	str	B3	false	false
							prixUnitaire	int			
							gamme	str			
FD4	B4	gamme	str	1	false	H4	categ	str	B4	false	false
FD5	B5	codeD	str	1	false	H5	mois	int	B5	false	false
							semestre	int			
							annee	int			
FD6	B6	codeClient	int	3	false	H6	montantV	int	B6	false	false
		codeProduit	int				qteV	int			
		codeD	str								

Figure 5.11 : Introduction de l'ensemble EnsDFN comme instances dans l'ontologie. XML-DFED_ONTO.

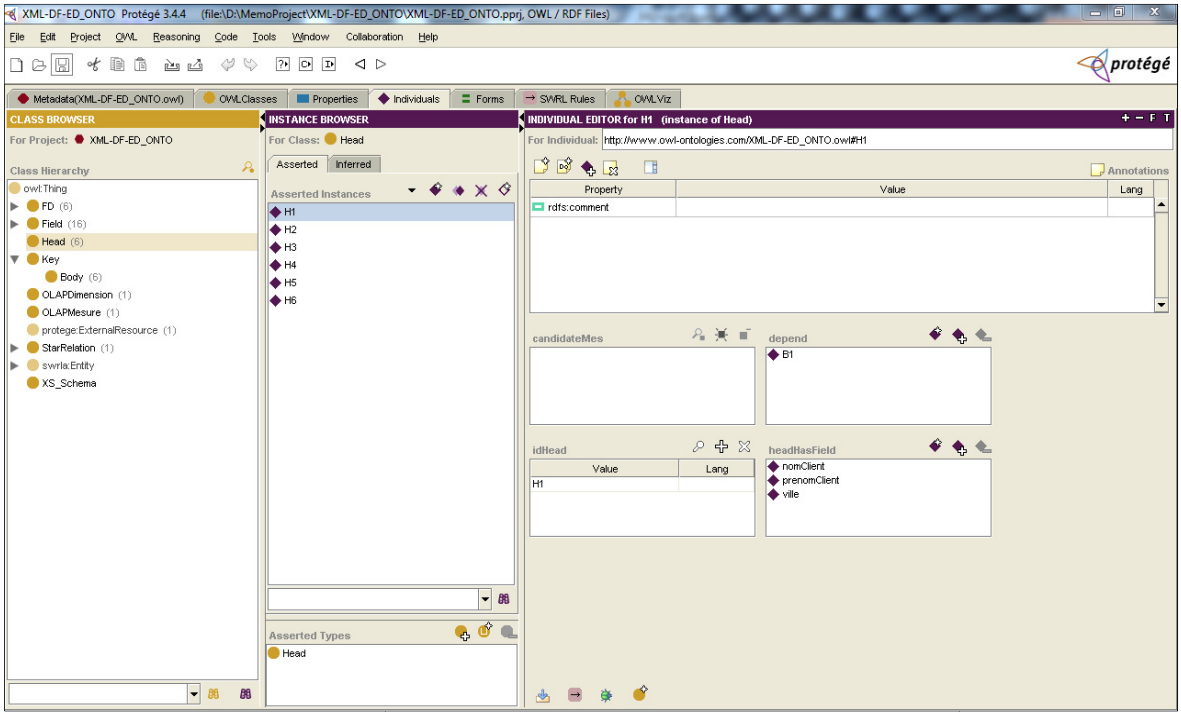


Figure 5.12 : Représentation d'une instance d'un résultat (Head) d'une dépendance fonctionnelle normalisée.

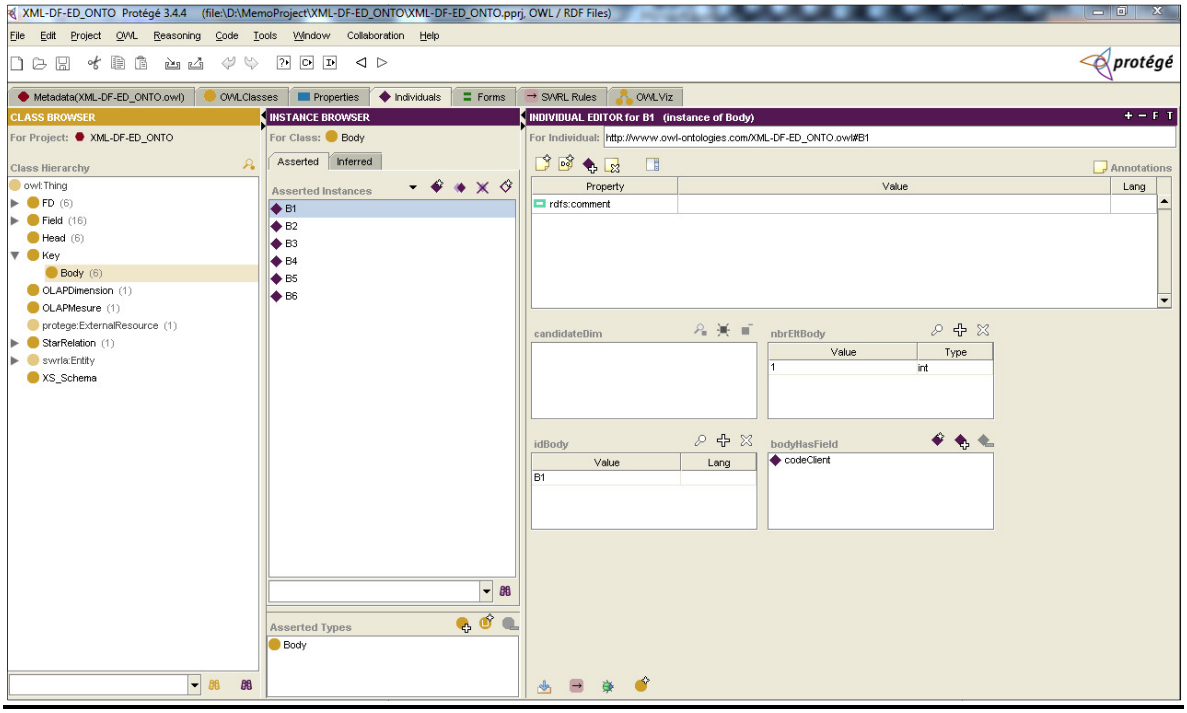


Figure 5.13 : Représentation d'une instance d'une prémisse (Body) d'une dépendance fonctionnelle normalisée.

Dans les figures 5.12 et 5.13, on présente l'instance FD1 (df1':

codeClient→nomClient, prenomClient, ville), cette dépendance fonctionnelle est composée d'une prémisse(Body) B1 et d'un résultat (Head) H1.

Dans ce qui suit, on présentera l'exécution des règles SWRL en les transformant en règles JESS, l'exécution se ferra avec le moteur JESS.

- Exécution de la règle « R1_searchMesCandidate »

Cette règle cherche les champs (Field) ayant un type entier, ces champs constitueront l'ensemble des mesures candidates.

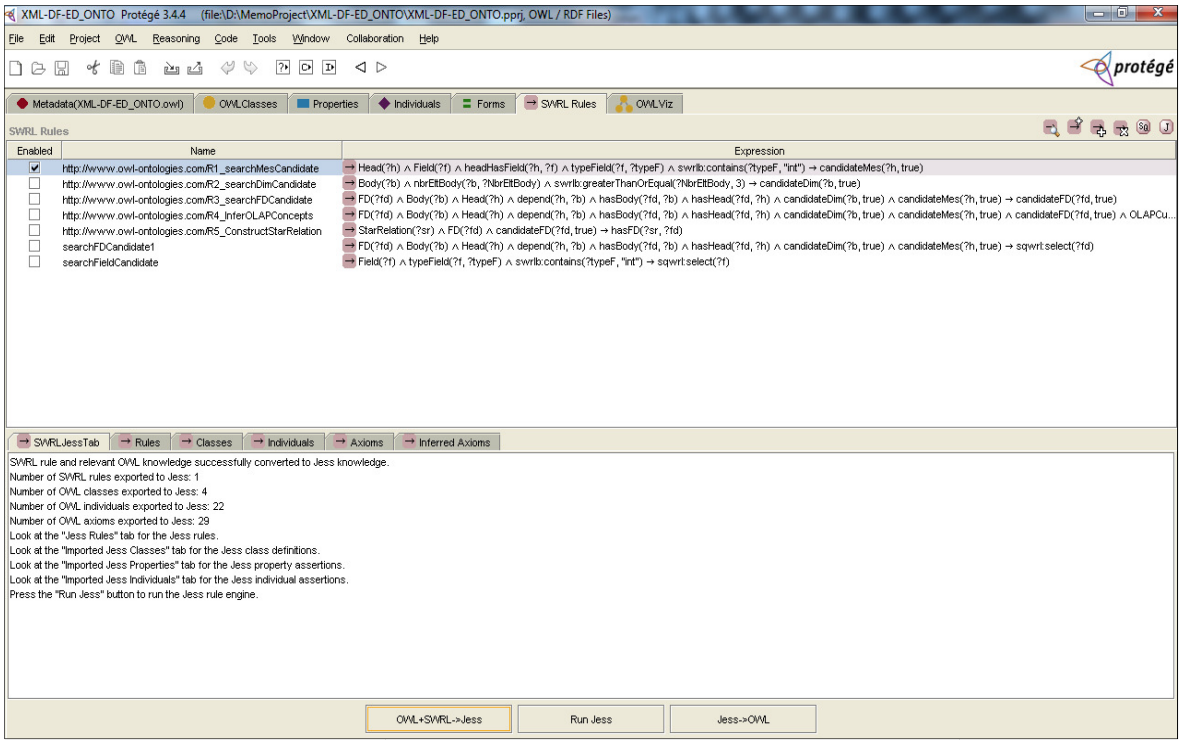


Figure 5.14 : Exécution de la règle «R1_searchMesCandidate».

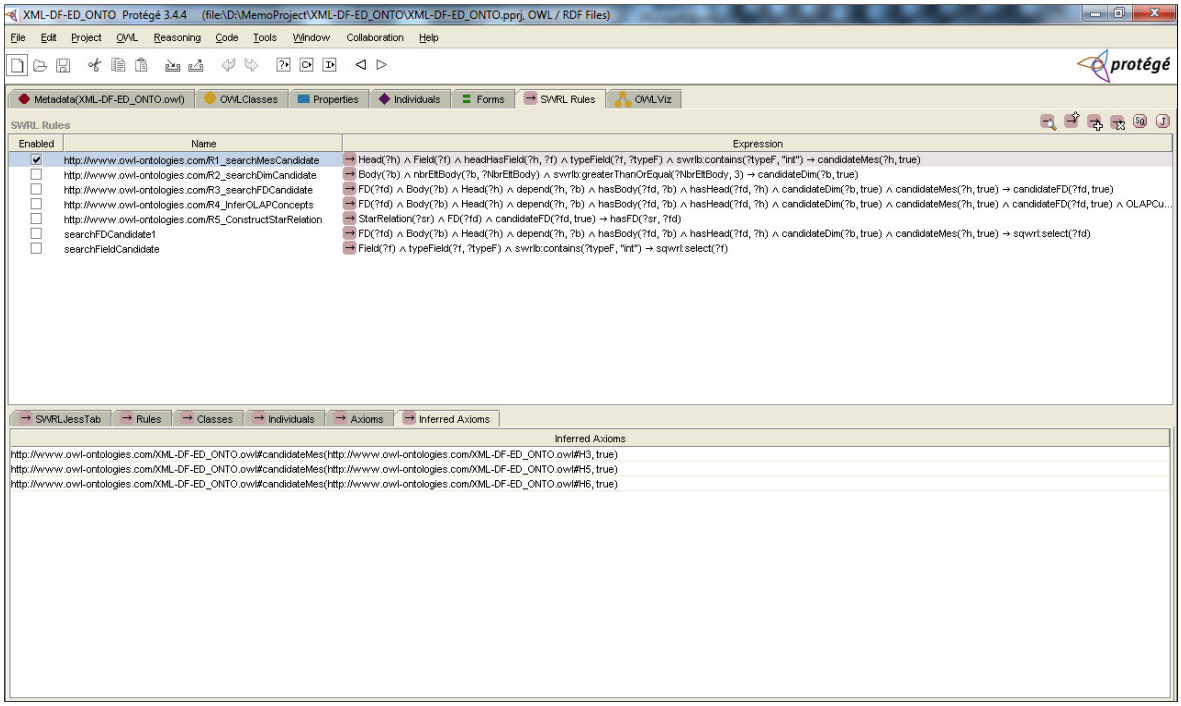


Figure 5.15 : Résultat de l'exécution de la règle «R1_searchMesCandidate».

On remarque que l'exécution de la règle « R1_searchMesCandidate» donne H3, H5 et H6 comme résultat car ils contiennent tous au moins un champ (Field) ayant un type entier.

- Exécution de la règle « R2_searchDimCandidate »

Cette règle repère parmi l'ensemble des prémisses (Body) celle qui a au minimum trois champs (Field). L'exécution donnera la prémisse B6 vu que cette dernière est la seule à contenir au moins trois champs (Field).

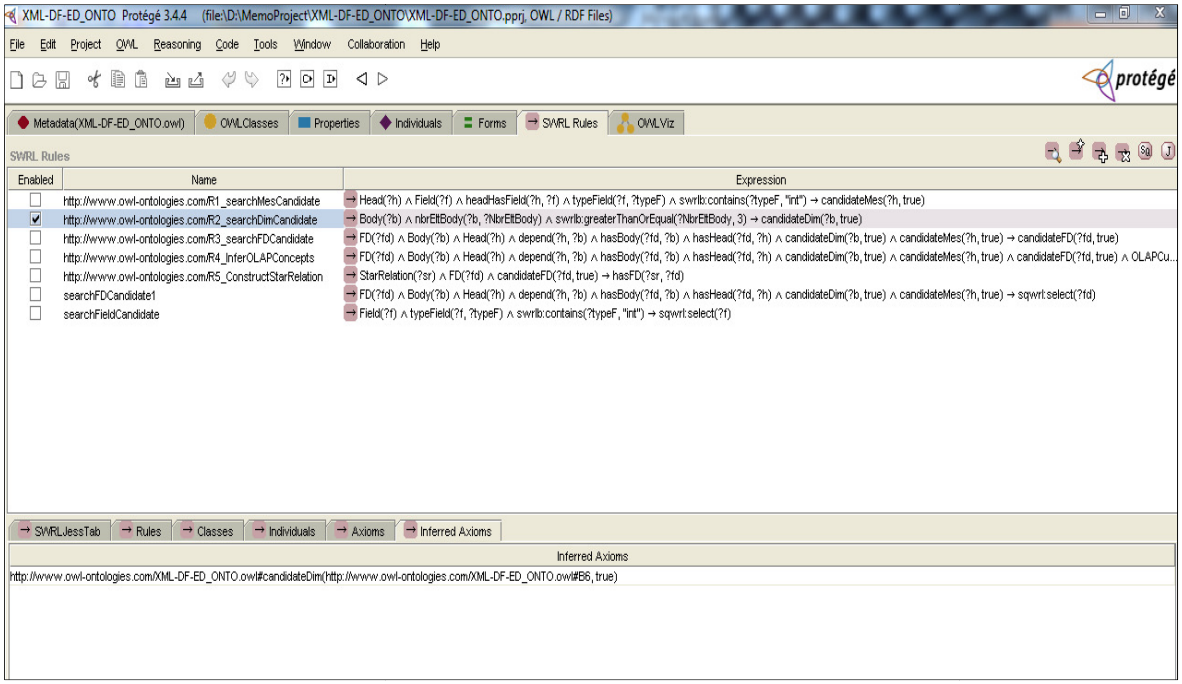


Figure 5.16 : Résultat de l'exécution de la règle «R2_searchDimCandidate».

- Exécution de la règle « R3_searchFDCandidate »

Cette règle détermine les dépendances fonctionnelles (FD) qui constitueront des cubes OLAP, elle se base sur la sélection des FD ayant une prémisse (Body) candidate et un résultat candidat (Head).

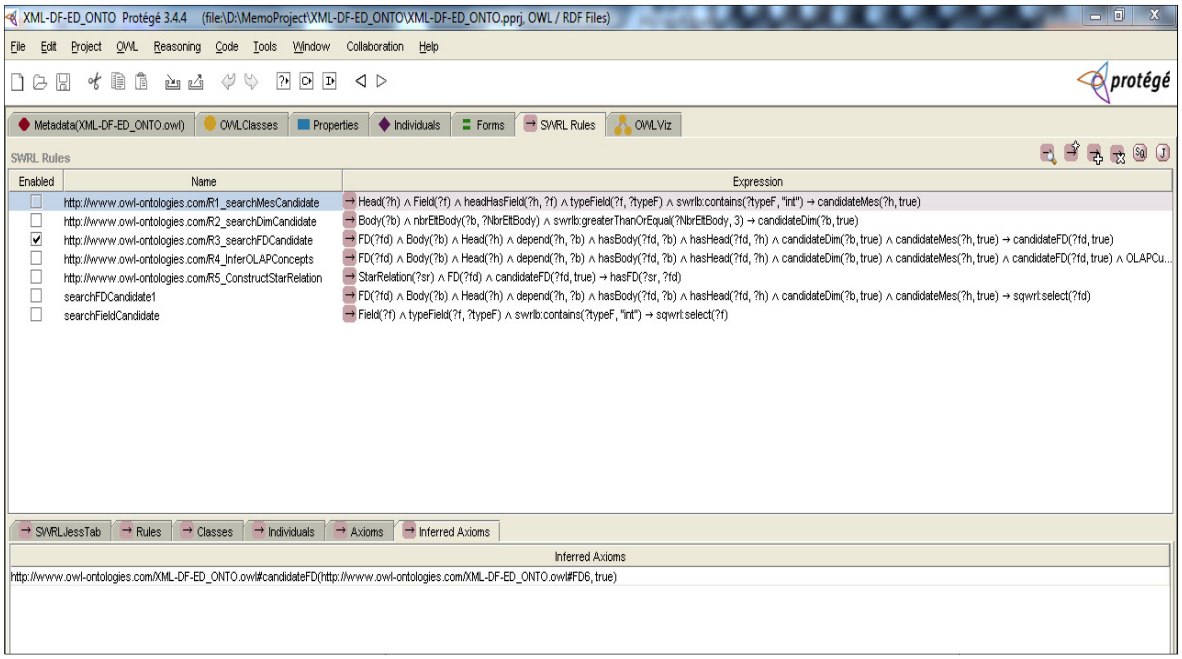


Figure 5.17 : Résultat de l'exécution de la règle «R3_searchFDCandidate».

➢ Exécution de la règle « R4_InferOLAPConcepts »

Cette règle déduit les concepts OLAP, elle se base sur les relations définis dans l'ontologie XML-DFED_ONTO.

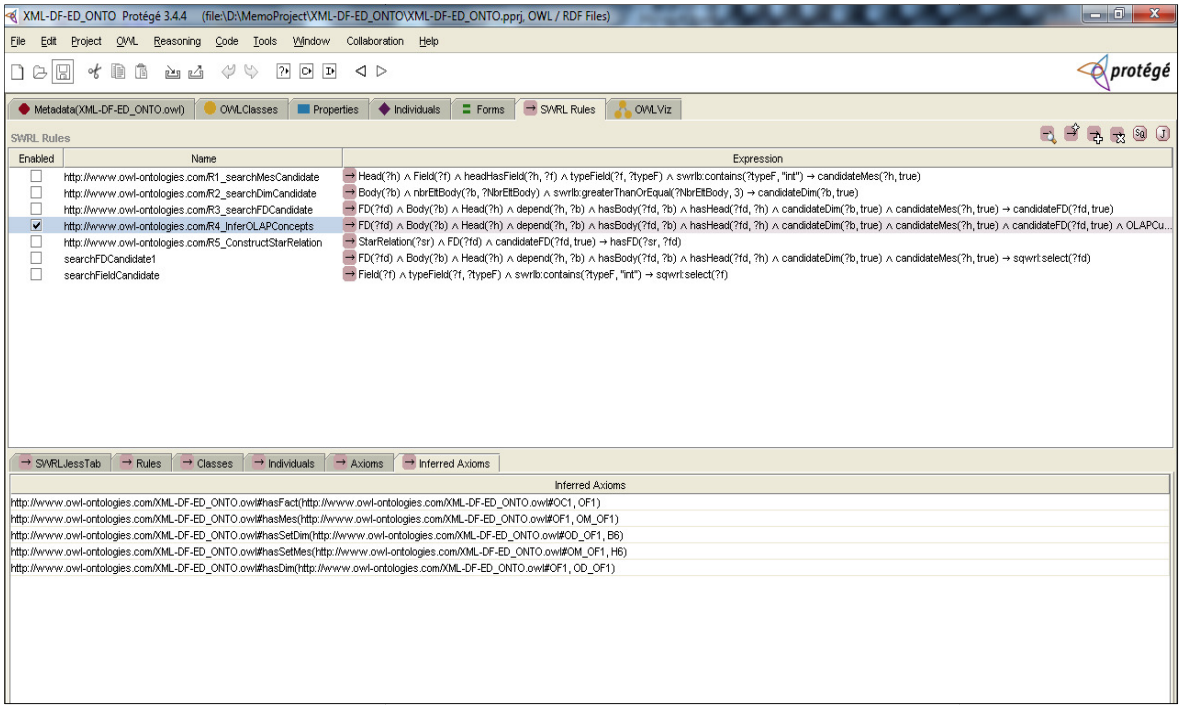


Figure 5.18 : Résultat de l'exécution de la règle «R4_inferOLAPConcepts».

➢ Exécution de la règle « R5_ConstructStarRelation»

Cette règle construit les relations en étoile et les cubes OLAP, elle construit à partir de chaque FD déduite avec la règle «R5_ConstructStarRelation» une relation en étoile ainsi qu'un cube OLAP.

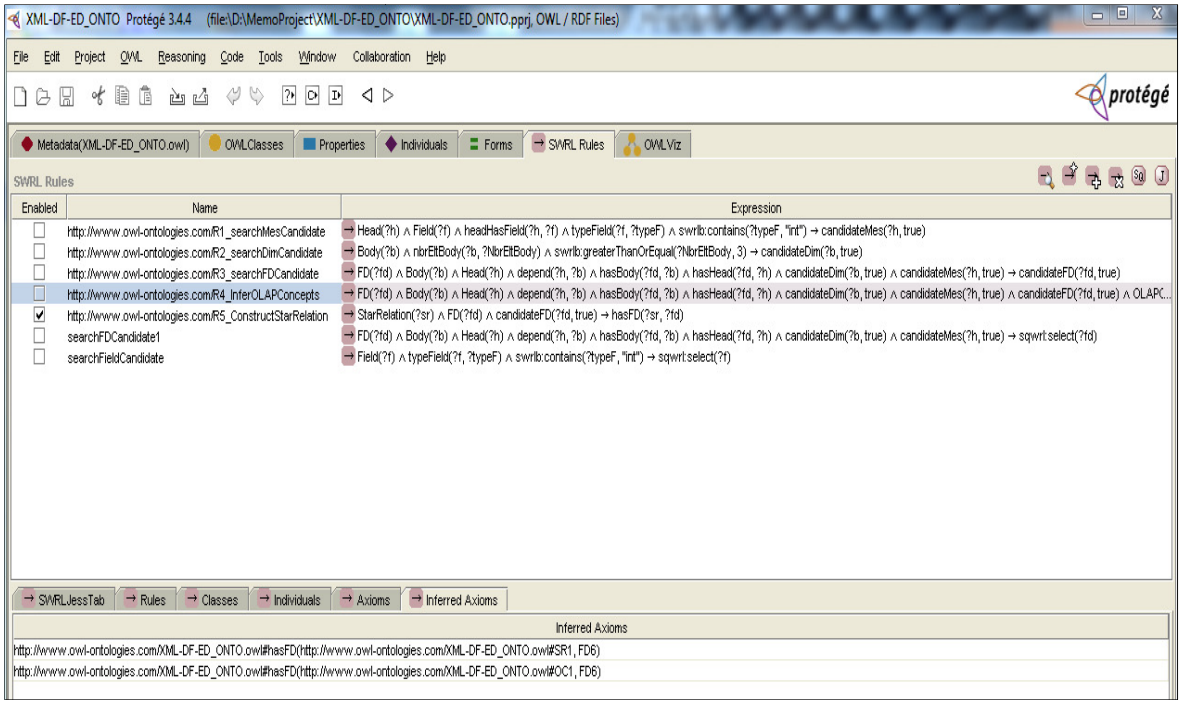


Figure 5.19 : Résultat de l'exécution de la règle «R5_ConstructStarRelation».

Nous avons présenté ci-dessus l'éxécution d'un test de conception d'un entrepot de données à partir d'un éxemple de schéma XML en utilisant notre approche XEDONTO qui est basée sur une ontologie de connaissances XML-DFED_ONTO. Ce test à permi de vérfier l'aspet fonctionnnel de notre approche.

5.4. Conclusion

Nous avons présenté dans ce chapitre une approche de conception automatisée d'un entrepôt de données XML basée sur une représentation des connaissances formalisée en une ontologie. On a exposé la définition de cette représentation des connaissances dans le langage naturel, puis nous avons formalisé et codé cette KR sous forme d'une ontologie avec le langage OWL. Nous avons enrichi celle-ci avec un ensemble de règles SWRL. Finalement, nous avons vérifié sa consistance avec le moteur d'inférence Pellet et nous avons déduit de nouvelles connaissances avec le moteur d'inférence JESS. En final, on a présenté un sénario d'éxécution de notre approche de conception XEDONTO pour démonter son aspect fonctionnel.

CONCLUSION GENERALE

Les systèmes d'information décisionnels (SID) sont devenus un thème de recherche important faisant réunir plusieurs disciplines (bases de données, statistiques, analyses de données, etc.) applicables à différents domaines (commerce, finance, médecine, e-commerce, …). Ils sont conçus autour d'entrepôts de données (ED). L'objectif des ED est de servir de support à la prise de décision en permettant une meilleure exploitation des informations contenues dans les systèmes opérationnels des entreprises. Cependant, leur mise en place est un travail de longue haleine, difficile et nécessite des acteurs avec des compétences non seulement assez variées, mais aussi complémentaires allant des bases de données, passant par la modélisation multidimensionnelle et touchant enfin, les techniques de spécification de besoins. Malgré les efforts déployés pas les chercheurs et les éditeurs de ces SID, des échecs de projets d'entreposage de données sont constatés. Ils sont dus à plusieurs raisons telles que la complexité de la conception et le manque de démarches appropriées **[114]**, le manque d'outils CASE d'aide à la spécification de besoins analytiques et de conception à partir d'une spécification, etc.

Suite à notre étude de l'état de l'art dans le domaine d'entreposage de données, nous avons relevé que les méthodes actuelles de conception des ED exigent soit une expertise dans les modèles des systèmes opérationnels (cas des approches ascendantes), soit une compétence dans l'expression des besoins analytiques (cas des approches descendantes), et ceci en plus de la maîtrise des modèles multidimensionnels. En pratique, les concepteurs se heurtent à plusieurs problèmes essentiellement dus à l'immaturité et à la complexité des méthodes de conception et à l'absence d'outils logiciels supportant ces méthodes. Nous nous sommes alors fixé l'objectif de proposer une approche d'aide à la conception automatisée d'un entrepôt de données à partir de schéma XML. Dans cette approche, nous visons à formaliser les règles d'automatisation recensées dans l'étude de l'état de l'art afin de construire une représentation de connaissances pour le concepteur et le logiciel, chose qui permettra de définir de manière rigoureuse les connaissances de conception d'un entrepôt de données.

1.Bilan des contributions

Dans ce mémoire nous avons présenté, dans une première partie, le contexte de nos travaux de recherche et dans une deuxième partie, une approche d'aide à la conception automatisée d'entrepôt de données en partant de schémas XML.

L'objectif de notre approche XEDONTO est de proposer au concepteur d'un entrepôt de donnée ainsi qu'au logiciel une représentation de connaissances regroupant les règles d'aide à la conception automatique d'un entrepôt de

données XML. Cette approche est une chaîne de deux phases : Elle débute par l'extraction des dépendances fonctionnelles d'un schéma XML puis la normalisation de cet ensemble avec l'algorithme d'Amstrong **[109]**. Ensuite, notre approche injecte cet ensemble de dépendances fonctionnelles normalisé dans l'ontologie XML-DFED_ONTO comme un ensemble d'instances et après inférence elle génère les concepts d'un schéma multidimensionnel.

Les apports de ce mémoire peuvent se résumer dans les points suivants :

- La proposition d'une représentation de connaissances sous forme d'une ontologie qui peut être utilisé par le concepteur et par le logiciel.

- La proposition d'une approche de conception guidée par les schémas XML.

- L'automatisation du processus de conception d'un entrepôt de données XML, à travers la génération automatique d'un schéma multidimentionnel.

2. Perspectives

Plusieurs perspectives de ce travail sont envisagées. En particulier :

- Etendre notre représentation de connaissances pour supporter les concepts du langage UML, vu que ce langage permettra au concepteur une compréhension rapide du schéma multidimensionnel résultant.

- Proposer d'autres règles de transformation d'un schéma XML vers un ensemble de dépendance fonctionnelle.

- Enrichir l'ensemble des règles SWRL en formalisant d'autres fonctions heuristiques permettant de repérer les concepts multidimensionnels.

- Implémenter le prototype de notre approche sous forme d'une bibliothèque pour une éventuelle réutilisation de la représentation de connaissances par les outils d'aide à la conception automatique d'un entrepôt de données XML.

ANNEXE
Script de l'ontologie XML-DFED_ONTO

```
<?xml version="1.0"?>
<rdf:RDF
    xmlns:xsp="http://www.owl-ontologies.com/2005/08/07/xsp.owl#"
    xmlns:swrlb="http://www.w3.org/2003/11/swrlb#"
    xmlns:swrla="http://swrl.stanford.edu/ontologies/3.3/swrla.owl#"
    xmlns:protege="http://protege.stanford.edu/plugins/owl/protege#"
    xmlns:swrl="http://www.w3.org/2003/11/swrl#"
    xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
    xmlns:xsd="http://www.w3.org/2001/XMLSchema#"
    xmlns:rdfs="http://www.w3.org/2000/01/rdf-schema#"
    xmlns:owl="http://www.w3.org/2002/07/owl#"
    xmlns:sqwrl="http://sqwrl.stanford.edu/ontologies/built-ins/3.4/sqwrl.owl#"
    xmlns="http://www.owl-ontologies.com/XML-DF-ED_ONTO.owl#"
    xmlns:assert="http://www.owl-ontologies.com/assert.owl#"
  xml:base="http://www.owl-ontologies.com/XML-DF-ED_ONTO.owl">
  <owl:Ontology rdf:about="">
    <owl:imports rdf:resource="http://sqwrl.stanford.edu/ontologies/built-ins/3.4/sqwrl.owl"/>
    <owl:imports rdf:resource="http://swrl.stanford.edu/ontologies/3.3/swrla.owl"/>
    <owl:imports rdf:resource="http://www.owl-ontologies.com/assert.owl"/>
  </owl:Ontology>
  <owl:Class rdf:ID="FD">
    <rdfs:subClassOf rdf:resource="http://www.w3.org/2002/07/owl#Thing"/>
    <rdfs:subClassOf>
      <owl:Restriction>
        <owl:onProperty>
          <owl:ObjectProperty rdf:ID="hasHead"/>
        </owl:onProperty>
        <owl:cardinality rdf:datatype="http://www.w3.org/2001/XMLSchema#int"
        >1</owl:cardinality>
      </owl:Restriction>
    </rdfs:subClassOf>
    <rdfs:subClassOf>
      <owl:Restriction>
        <owl:cardinality rdf:datatype="http://www.w3.org/2001/XMLSchema#int"
        >1</owl:cardinality>
        <owl:onProperty>
          <owl:ObjectProperty rdf:ID="hasBody"/>
        </owl:onProperty>
      </owl:Restriction>
    </rdfs:subClassOf>
  </owl:Class>
  <owl:Class rdf:ID="Head"/>
  <owl:Class rdf:ID="XS_Element">
    <rdfs:subClassOf>
      <owl:Class rdf:ID="Field"/>
    </rdfs:subClassOf>
  </owl:Class>
```

```
<owl:Class rdf:ID="XS_Schema"/>
<owl:Class rdf:ID="OLAPFact">
  <rdfs:subClassOf rdf:resource="#FD"/>
</owl:Class>
<owl:Class rdf:ID="StarRelation">
  <rdfs:comment rdf:datatype="http://www.w3.org/2001/XMLSchema#string"
  ></rdfs:comment>
  <rdfs:subClassOf rdf:resource="http://www.w3.org/2002/07/owl#Thing"/>
  <rdfs:subClassOf>
    <owl:Restriction>
      <owl:minCardinality rdf:datatype="http://www.w3.org/2001/XMLSchema#int"
      >1</owl:minCardinality>
      <owl:onProperty>
        <owl:ObjectProperty rdf:ID="hasFD"/>
      </owl:onProperty>
    </owl:Restriction>
  </rdfs:subClassOf>
</owl:Class>
<owl:Class rdf:ID="OLAPCube">
  <rdfs:subClassOf rdf:resource="#StarRelation"/>
</owl:Class>
<owl:Class rdf:ID="OLAPMesure"/>
<owl:Class rdf:ID="OLAPDimension"/>
<owl:Class rdf:ID="Body">
  <rdfs:subClassOf>
    <owl:Class rdf:ID="Key"/>
  </rdfs:subClassOf>
</owl:Class>
<owl:Class rdf:ID="XS_Complex_Element">
  <rdfs:subClassOf rdf:resource="#FD"/>
</owl:Class>
<owl:ObjectProperty rdf:ID="depend">
  <rdfs:domain rdf:resource="#Head"/>
  <rdfs:range rdf:resource="#Body"/>
</owl:ObjectProperty>
<owl:ObjectProperty rdf:ID="hasMes">
  <rdfs:domain rdf:resource="#OLAPFact"/>
  <rdfs:range rdf:resource="#OLAPMesure"/>
</owl:ObjectProperty>
<owl:ObjectProperty rdf:about="#hasHead">
  <rdfs:range rdf:resource="#Head"/>
  <rdfs:domain rdf:resource="#FD"/>
</owl:ObjectProperty>
<owl:ObjectProperty rdf:ID="hasSetDim">
  <rdfs:range rdf:resource="#Body"/>
  <rdfs:domain rdf:resource="#OLAPDimension"/>
</owl:ObjectProperty>
<owl:ObjectProperty rdf:ID="bodyHasField">
  <rdfs:domain rdf:resource="#Body"/>
  <rdfs:range rdf:resource="#Field"/>
</owl:ObjectProperty>
<owl:ObjectProperty rdf:ID="hasFact">
  <rdfs:range rdf:resource="#OLAPFact"/>
  <rdfs:domain rdf:resource="#OLAPCube"/>
</owl:ObjectProperty>
<owl:ObjectProperty rdf:ID="headHasField">
  <rdfs:domain rdf:resource="#Head"/>
  <rdfs:range rdf:resource="#Field"/>
</owl:ObjectProperty>
<owl:ObjectProperty rdf:ID="schemaHasCpxElt">
```

```
  <rdfs:domain rdf:resource="#XS_Schema"/>
  <rdfs:range rdf:resource="#XS_Complex_Element"/>
</owl:ObjectProperty>
<owl:ObjectProperty rdf:ID="hasSetMes">
  <rdfs:range rdf:resource="#Head"/>
  <rdfs:domain rdf:resource="#OLAPMesure"/>
</owl:ObjectProperty>
<owl:ObjectProperty rdf:ID="hasElt">
  <rdfs:range rdf:resource="#XS_Element"/>
  <rdfs:domain rdf:resource="#XS_Complex_Element"/>
</owl:ObjectProperty>
<owl:ObjectProperty rdf:ID="hasDim">
  <rdfs:range rdf:resource="#OLAPDimension"/>
  <rdfs:domain rdf:resource="#OLAPFact"/>
</owl:ObjectProperty>
<owl:ObjectProperty rdf:about="#hasBody">
  <rdfs:domain rdf:resource="#FD"/>
  <rdfs:range rdf:resource="#Body"/>
</owl:ObjectProperty>
<owl:ObjectProperty rdf:about="#hasFD">
  <rdfs:domain rdf:resource="#StarRelation"/>
  <rdfs:range rdf:resource="#FD"/>
</owl:ObjectProperty>
<owl:DatatypeProperty rdf:ID="candidateDim">
  <rdfs:range>
    <owl:DataRange>
      <owl:oneOf rdf:parseType="Resource">
        <rdf:first rdf:datatype="http://www.w3.org/2001/XMLSchema#boolean"
        >true</rdf:first>
        <rdf:rest rdf:parseType="Resource">
          <rdf:first rdf:datatype="http://www.w3.org/2001/XMLSchema#boolean"
          >false</rdf:first>
          <rdf:rest rdf:resource="http://www.w3.org/1999/02/22-rdf-syntax-ns#nil"/>
        </rdf:rest>
      </owl:oneOf>
    </owl:DataRange>
  </rdfs:range>
  <rdfs:domain rdf:resource="#Body"/>
</owl:DatatypeProperty>
<owl:DatatypeProperty rdf:ID="candidateFD">
  <rdfs:domain rdf:resource="#FD"/>
  <rdfs:range>
    <owl:DataRange>
      <owl:oneOf rdf:parseType="Resource">
        <rdf:rest rdf:parseType="Resource">
          <rdf:first rdf:datatype="http://www.w3.org/2001/XMLSchema#boolean"
          >false</rdf:first>
          <rdf:rest rdf:resource="http://www.w3.org/1999/02/22-rdf-syntax-ns#nil"/>
        </rdf:rest>
        <rdf:first rdf:datatype="http://www.w3.org/2001/XMLSchema#boolean"
        >true</rdf:first>
      </owl:oneOf>
    </owl:DataRange>
  </rdfs:range>
</owl:DatatypeProperty>
<owl:DatatypeProperty rdf:ID="idField">
  <rdfs:range rdf:resource="http://www.w3.org/2001/XMLSchema#string"/>
  <rdfs:domain rdf:resource="#Field"/>
</owl:DatatypeProperty>
<owl:DatatypeProperty rdf:ID="idBody">
```

```
  <rdfs:range rdf:resource="http://www.w3.org/2001/XMLSchema#string"/>
  <rdfs:domain rdf:resource="#Body"/>
</owl:DatatypeProperty>
<owl:DatatypeProperty rdf:ID="idHead">
  <rdfs:range rdf:resource="http://www.w3.org/2001/XMLSchema#string"/>
  <rdfs:domain rdf:resource="#Head"/>
</owl:DatatypeProperty>
<owl:DatatypeProperty rdf:ID="nbrEltBody">
  <rdfs:range rdf:resource="http://www.w3.org/2001/XMLSchema#int"/>
  <rdfs:domain rdf:resource="#Body"/>
</owl:DatatypeProperty>
<owl:DatatypeProperty rdf:ID="idRel">
  <rdfs:range rdf:resource="http://www.w3.org/2001/XMLSchema#string"/>
  <rdfs:domain rdf:resource="#StarRelation"/>
</owl:DatatypeProperty>
<owl:DatatypeProperty rdf:ID="candidateMes">
  <rdfs:range>
    <owl:DataRange>
      <owl:oneOf rdf:parseType="Resource">
        <rdf:first rdf:datatype="http://www.w3.org/2001/XMLSchema#boolean"
        >true</rdf:first>
        <rdf:rest rdf:parseType="Resource">
          <rdf:rest rdf:resource="http://www.w3.org/1999/02/22-rdf-syntax-ns#nil"/>
          <rdf:first rdf:datatype="http://www.w3.org/2001/XMLSchema#boolean"
          >false</rdf:first>
        </rdf:rest>
      </owl:oneOf>
    </owl:DataRange>
  </rdfs:range>
  <rdfs:domain rdf:resource="#Head"/>
</owl:DatatypeProperty>
<owl:DatatypeProperty rdf:ID="nameXS_Element">
  <rdfs:domain rdf:resource="#XS_Element"/>
  <rdfs:range rdf:resource="http://www.w3.org/2001/XMLSchema#string"/>
</owl:DatatypeProperty>
<owl:DatatypeProperty rdf:ID="typeField">
  <rdfs:range rdf:resource="http://www.w3.org/2001/XMLSchema#string"/>
  <rdfs:domain rdf:resource="#Field"/>
</owl:DatatypeProperty>
<owl:DatatypeProperty rdf:ID="idFD">
  <rdfs:domain rdf:resource="#FD"/>
  <rdfs:range rdf:resource="http://www.w3.org/2001/XMLSchema#string"/>
</owl:DatatypeProperty>
<owl:DatatypeProperty rdf:ID="idSchema">
  <rdfs:range rdf:resource="http://www.w3.org/2001/XMLSchema#string"/>
  <rdfs:domain rdf:resource="#XS_Schema"/>
</owl:DatatypeProperty>
<swrl:Variable rdf:ID="g"/>
<swrl:Imp rdf:about="http://www.owl-ontologies.com/R5_ConstructStarRelation">
  <swrl:head>
    <swrl:AtomList>
      <rdf:rest rdf:resource="http://www.w3.org/1999/02/22-rdf-syntax-ns#nil"/>
      <rdf:first>
        <swrl:IndividualPropertyAtom>
          <swrl:argument2>
            <swrl:Variable rdf:ID="fd"/>
          </swrl:argument2>
          <swrl:propertyPredicate rdf:resource="#hasFD"/>
          <swrl:argument1>
            <swrl:Variable rdf:ID="sr"/>
```

```
        </swrl:argument1>
       </swrl:IndividualPropertyAtom>
      </rdf:first>
    </swrl:AtomList>
  </swrl:head>
  <swrla:isRuleEnabled rdf:datatype="http://www.w3.org/2001/XMLSchema#boolean"
  >false</swrla:isRuleEnabled>
  <swrl:body>
    <swrl:AtomList>
      <rdf:rest>
        <swrl:AtomList>
          <rdf:first>
            <swrl:ClassAtom>
              <swrl:classPredicate rdf:resource="#FD"/>
              <swrl:argument1 rdf:resource="#fd"/>
            </swrl:ClassAtom>
          </rdf:first>
          <rdf:rest>
            <swrl:AtomList>
              <rdf:first>
                <swrl:DatavaluedPropertyAtom>
                  <swrl:propertyPredicate rdf:resource="#candidateFD"/>
                  <swrl:argument2 rdf:datatype="http://www.w3.org/2001/XMLSchema#boolean"
                  >true</swrl:argument2>
                  <swrl:argument1 rdf:resource="#fd"/>
                </swrl:DatavaluedPropertyAtom>
              </rdf:first>
              <rdf:rest rdf:resource="http://www.w3.org/1999/02/22-rdf-syntax-ns#nil"/>
            </swrl:AtomList>
          </rdf:rest>
        </swrl:AtomList>
      </rdf:rest>
      <rdf:first>
        <swrl:ClassAtom>
          <swrl:classPredicate rdf:resource="#StarRelation"/>
          <swrl:argument1 rdf:resource="#sr"/>
        </swrl:ClassAtom>
      </rdf:first>
    </swrl:AtomList>
  </swrl:body>
</swrl:Imp>
<swrl:Imp rdf:about="http://www.owl-ontologies.com/R1_searchMesCandidate">
  <swrl:head>
    <swrl:AtomList>
      <rdf:rest rdf:resource="http://www.w3.org/1999/02/22-rdf-syntax-ns#nil"/>
      <rdf:first>
        <swrl:DatavaluedPropertyAtom>
          <swrl:propertyPredicate rdf:resource="#candidateMes"/>
          <swrl:argument1>
            <swrl:Variable rdf:ID="h"/>
          </swrl:argument1>
          <swrl:argument2 rdf:datatype="http://www.w3.org/2001/XMLSchema#boolean"
          >true</swrl:argument2>
        </swrl:DatavaluedPropertyAtom>
      </rdf:first>
    </swrl:AtomList>
  </swrl:head>
  <swrla:isRuleEnabled rdf:datatype="http://www.w3.org/2001/XMLSchema#boolean"
  >false</swrla:isRuleEnabled>
  <swrl:body>
```

```
<swrl:AtomList>
 <rdf:rest>
  <swrl:AtomList>
   <rdf:rest>
    <swrl:AtomList>
     <rdf:first>
      <swrl:IndividualPropertyAtom>
       <swrl:argument1 rdf:resource="#h"/>
       <swrl:propertyPredicate rdf:resource="#headHasField"/>
       <swrl:argument2>
        <swrl:Variable rdf:ID="f"/>
       </swrl:argument2>
      </swrl:IndividualPropertyAtom>
     </rdf:first>
     <rdf:rest>
      <swrl:AtomList>
       <rdf:first>
        <swrl:DatavaluedPropertyAtom>
         <swrl:propertyPredicate rdf:resource="#typeField"/>
         <swrl:argument1 rdf:resource="#f"/>
         <swrl:argument2>
          <swrl:Variable rdf:ID="typeF"/>
         </swrl:argument2>
        </swrl:DatavaluedPropertyAtom>
       </rdf:first>
       <rdf:rest>
        <swrl:AtomList>
         <rdf:rest rdf:resource="http://www.w3.org/1999/02/22-rdf-syntax-ns#nil"/>
         <rdf:first>
          <swrl:BuiltinAtom>
           <swrl:arguments>
            <rdf:List>
             <rdf:first rdf:resource="#typeF"/>
             <rdf:rest>
              <rdf:List>
               <rdf:first rdf:datatype="http://www.w3.org/2001/XMLSchema#string"
               >int</rdf:first>
               <rdf:rest rdf:resource="http://www.w3.org/1999/02/22-rdf-syntax-ns#nil"/>
              </rdf:List>
             </rdf:rest>
            </rdf:List>
           </swrl:arguments>
           <swrl:builtin rdf:resource="http://www.w3.org/2003/11/swrlb#contains"/>
          </swrl:BuiltinAtom>
         </rdf:first>
        </swrl:AtomList>
       </rdf:rest>
      </swrl:AtomList>
     </rdf:rest>
    </swrl:AtomList>
   </rdf:rest>
   <rdf:first>
    <swrl:ClassAtom>
     <swrl:argument1 rdf:resource="#f"/>
     <swrl:classPredicate rdf:resource="#Field"/>
    </swrl:ClassAtom>
   </rdf:first>
  </swrl:AtomList>
 </rdf:rest>
 <rdf:first>
```

```
        <swrl:ClassAtom>
          <swrl:classPredicate rdf:resource="#Head"/>
          <swrl:argument1 rdf:resource="#h"/>
        </swrl:ClassAtom>
      </rdf:first>
    </swrl:AtomList>
  </swrl:body>
 </swrl:Imp>
 <rdf:Description rdf:ID="Field_3">
  <idField rdf:datatype="http://www.w3.org/2001/XMLSchema#string"
  ></idField>
 </rdf:Description>
 <swrl:Imp rdf:ID="searchFDCandidate1">
  <swrla:isRuleEnabled rdf:datatype="http://www.w3.org/2001/XMLSchema#boolean"
  >true</swrla:isRuleEnabled>
  <swrl:head>
   <swrl:AtomList>
    <rdf:first>
     <swrl:BuiltinAtom>
      <swrl:arguments>
       <rdf:List>
        <rdf:first rdf:resource="#fd"/>
        <rdf:rest rdf:resource="http://www.w3.org/1999/02/22-rdf-syntax-ns#nil"/>
       </rdf:List>
      </swrl:arguments>
      <swrl:builtin                    rdf:resource="http://sqwrl.stanford.edu/ontologies/built-
ins/3.4/sqwrl.owl#select"/>
     </swrl:BuiltinAtom>
    </rdf:first>
    <rdf:rest rdf:resource="http://www.w3.org/1999/02/22-rdf-syntax-ns#nil"/>
   </swrl:AtomList>
  </swrl:head>
  <swrl:body>
   <swrl:AtomList>
    <rdf:rest>
     <swrl:AtomList>
      <rdf:rest>
       <swrl:AtomList>
        <rdf:rest>
         <swrl:AtomList>
          <rdf:rest>
           <swrl:AtomList>
            <rdf:rest>
             <swrl:AtomList>
              <rdf:first>
               <swrl:IndividualPropertyAtom>
                <swrl:argument1 rdf:resource="#fd"/>
                <swrl:propertyPredicate rdf:resource="#hasHead"/>
                <swrl:argument2 rdf:resource="#h"/>
               </swrl:IndividualPropertyAtom>
              </rdf:first>
              <rdf:rest>
               <swrl:AtomList>
                <rdf:rest>
                 <swrl:AtomList>
                  <rdf:first>
                   <swrl:DatavaluedPropertyAtom>
                    <swrl:argument2 rdf:datatype=
                    "http://www.w3.org/2001/XMLSchema#boolean"
                    >true</swrl:argument2>
```

```
                    <swrl:propertyPredicate rdf:resource="#candidateMes"/>
                    <swrl:argument1 rdf:resource="#h"/>
                   </swrl:DatavaluedPropertyAtom>
                  </rdf:first>
                  <rdf:rest rdf:resource="http://www.w3.org/1999/02/22-rdf-syntax-ns#nil"/>
                 </swrl:AtomList>
                </rdf:rest>
                <rdf:first>
                 <swrl:DatavaluedPropertyAtom>
                  <swrl:argument2 rdf:datatype=
                  "http://www.w3.org/2001/XMLSchema#boolean"
                  >true</swrl:argument2>
                  <swrl:argument1>
                   <swrl:Variable rdf:ID="b"/>
                  </swrl:argument1>
                  <swrl:propertyPredicate rdf:resource="#candidateDim"/>
                 </swrl:DatavaluedPropertyAtom>
                </rdf:first>
               </swrl:AtomList>
              </rdf:rest>
             </swrl:AtomList>
            </rdf:rest>
            <rdf:first>
             <swrl:IndividualPropertyAtom>
              <swrl:propertyPredicate rdf:resource="#hasBody"/>
              <swrl:argument1 rdf:resource="#fd"/>
              <swrl:argument2 rdf:resource="#b"/>
             </swrl:IndividualPropertyAtom>
            </rdf:first>
           </swrl:AtomList>
          </rdf:rest>
          <rdf:first>
           <swrl:IndividualPropertyAtom>
            <swrl:propertyPredicate rdf:resource="#depend"/>
            <swrl:argument2 rdf:resource="#b"/>
            <swrl:argument1 rdf:resource="#h"/>
           </swrl:IndividualPropertyAtom>
          </rdf:first>
         </swrl:AtomList>
        </rdf:rest>
        <rdf:first>
         <swrl:ClassAtom>
          <swrl:classPredicate rdf:resource="#Head"/>
          <swrl:argument1 rdf:resource="#h"/>
         </swrl:ClassAtom>
        </rdf:first>
       </swrl:AtomList>
      </rdf:rest>
      <rdf:first>
       <swrl:ClassAtom>
        <swrl:argument1 rdf:resource="#b"/>
        <swrl:classPredicate rdf:resource="#Body"/>
       </swrl:ClassAtom>
      </rdf:first>
     </swrl:AtomList>
    </rdf:rest>
    <rdf:first>
     <swrl:ClassAtom>
      <swrl:classPredicate rdf:resource="#FD"/>
      <swrl:argument1 rdf:resource="#fd"/>
```

```
        </swrl:ClassAtom>
      </rdf:first>
    </swrl:AtomList>
  </swrl:body>
</swrl:Imp>
<FD rdf:ID="FD3">
  <idFD rdf:datatype="http://www.w3.org/2001/XMLSchema#string"
  >FD3</idFD>
  <hasHead>
    <Head rdf:ID="H3">
      <depend>
        <Body rdf:ID="B3">
          <nbrEltBody rdf:datatype="http://www.w3.org/2001/XMLSchema#int"
          >1</nbrEltBody>
          <bodyHasField>
            <Field rdf:ID="codeProduit">
              <typeField rdf:datatype="http://www.w3.org/2001/XMLSchema#string"
              >int</typeField>
              <idField rdf:datatype="http://www.w3.org/2001/XMLSchema#string"
              >codeProduit</idField>
            </Field>
          </bodyHasField>
          <idBody rdf:datatype="http://www.w3.org/2001/XMLSchema#string"
          >B3</idBody>
        </Body>
      </depend>
      <idHead rdf:datatype="http://www.w3.org/2001/XMLSchema#string"
      >H3</idHead>
      <headHasField>
        <Field rdf:ID="gamme">
          <idField rdf:datatype="http://www.w3.org/2001/XMLSchema#string"
          >gamme</idField>
          <typeField rdf:datatype="http://www.w3.org/2001/XMLSchema#string"
          >str</typeField>
        </Field>
      </headHasField>
      <headHasField>
        <Field rdf:ID="prixUnitaire">
          <idField rdf:datatype="http://www.w3.org/2001/XMLSchema#string"
          >prixUnitaire</idField>
          <typeField rdf:datatype="http://www.w3.org/2001/XMLSchema#string"
          >int</typeField>
        </Field>
      </headHasField>
      <headHasField>
        <Field rdf:ID="description">
          <idField rdf:datatype="http://www.w3.org/2001/XMLSchema#string"
          >description</idField>
          <typeField rdf:datatype="http://www.w3.org/2001/XMLSchema#string"
          >str</typeField>
        </Field>
      </headHasField>
      <candidateMes rdf:datatype="http://www.w3.org/2001/XMLSchema#boolean"
      >true</candidateMes>
    </Head>
  </hasHead>
  <hasBody rdf:resource="#B3"/>
</FD>
<FD rdf:ID="FD2">
  <hasHead>
```

```
    <Head rdf:ID="H2">
      <headHasField>
        <Field rdf:ID="pays">
          <typeField rdf:datatype="http://www.w3.org/2001/XMLSchema#string"
          >str</typeField>
          <idField rdf:datatype="http://www.w3.org/2001/XMLSchema#string"
          >pays</idField>
        </Field>
      </headHasField>
      <depend>
        <Body rdf:ID="B2">
          <nbrEltBody rdf:datatype="http://www.w3.org/2001/XMLSchema#int"
          >1</nbrEltBody>
          <idBody rdf:datatype="http://www.w3.org/2001/XMLSchema#string"
          >B2</idBody>
          <bodyHasField>
            <Field rdf:ID="ville">
              <typeField rdf:datatype="http://www.w3.org/2001/XMLSchema#string"
              >str</typeField>
              <idField rdf:datatype="http://www.w3.org/2001/XMLSchema#string"
              >ville</idField>
            </Field>
          </bodyHasField>
        </Body>
      </depend>
      <idHead rdf:datatype="http://www.w3.org/2001/XMLSchema#string"
      >H2</idHead>
    </Head>
  </hasHead>
  <hasBody rdf:resource="#B2"/>
  <idFD rdf:datatype="http://www.w3.org/2001/XMLSchema#string"
  >FD2</idFD>
</FD>
<FD rdf:ID="FD1">
  <hasHead>
    <Head rdf:ID="H1">
      <idHead rdf:datatype="http://www.w3.org/2001/XMLSchema#string"
      >H1</idHead>
      <headHasField rdf:resource="#ville"/>
      <headHasField>
        <Field rdf:ID="prenomClient">
          <typeField rdf:datatype="http://www.w3.org/2001/XMLSchema#string"
          >str</typeField>
          <idField rdf:datatype="http://www.w3.org/2001/XMLSchema#string"
          >prenomClient</idField>
        </Field>
      </headHasField>
      <headHasField>
        <Field rdf:ID="nomClient">
          <typeField rdf:datatype="http://www.w3.org/2001/XMLSchema#string"
          >str</typeField>
          <idField rdf:datatype="http://www.w3.org/2001/XMLSchema#string"
          >nomClient</idField>
        </Field>
      </headHasField>
      <depend>
        <Body rdf:ID="B1">
          <nbrEltBody rdf:datatype="http://www.w3.org/2001/XMLSchema#int"
          >1</nbrEltBody>
          <idBody rdf:datatype="http://www.w3.org/2001/XMLSchema#string"
```

```
        >B1</idBody>
        <bodyHasField>
          <Field rdf:ID="codeClient">
            <typeField rdf:datatype="http://www.w3.org/2001/XMLSchema#string"
            >int</typeField>
            <idField rdf:datatype="http://www.w3.org/2001/XMLSchema#string"
            >codeClient</idField>
          </Field>
        </bodyHasField>
      </Body>
    </depend>
  </Head>
 </hasHead>
 <hasBody rdf:resource="#B1"/>
 <idFD rdf:datatype="http://www.w3.org/2001/XMLSchema#string"
 >FD1</idFD>
</FD>
<OLAPCube rdf:ID="OC1">
 <hasFact>
  <OLAPFact rdf:ID="OF1">
   <hasMes>
    <OLAPMesure rdf:ID="OM_OF1">
     <hasSetMes>
      <Head rdf:ID="H6">
       <depend>
        <Body rdf:ID="B6">
         <idBody rdf:datatype="http://www.w3.org/2001/XMLSchema#string"
         >B6</idBody>
         <nbrEltBody rdf:datatype="http://www.w3.org/2001/XMLSchema#int"
         >3</nbrEltBody>
         <bodyHasField>
          <Field rdf:ID="codeD">
           <typeField rdf:datatype="http://www.w3.org/2001/XMLSchema#string"
           >str</typeField>
           <idField rdf:datatype="http://www.w3.org/2001/XMLSchema#string"
           >CodeD</idField>
          </Field>
         </bodyHasField>
         <bodyHasField rdf:resource="#codeProduit"/>
         <bodyHasField rdf:resource="#codeClient"/>
         <candidateDim rdf:datatype="http://www.w3.org/2001/XMLSchema#boolean"
         >true</candidateDim>
        </Body>
       </depend>
       <idHead rdf:datatype="http://www.w3.org/2001/XMLSchema#string"
       >H6</idHead>
       <headHasField>
        <Field rdf:ID="qteV">
         <typeField rdf:datatype="http://www.w3.org/2001/XMLSchema#string"
         >int</typeField>
         <idField rdf:datatype="http://www.w3.org/2001/XMLSchema#string"
         >qteV</idField>
        </Field>
       </headHasField>
       <headHasField>
        <Field rdf:ID="montantV">
         <idField rdf:datatype="http://www.w3.org/2001/XMLSchema#string"
         >montantV</idField>
         <typeField rdf:datatype="http://www.w3.org/2001/XMLSchema#string"
         >int</typeField>
```

```
            </Field>
           </headHasField>
           <candidateMes rdf:datatype="http://www.w3.org/2001/XMLSchema#boolean"
           >true</candidateMes>
          </Head>
         </hasSetMes>
        </OLAPMesure>
       </hasMes>
       <idFD rdf:datatype="http://www.w3.org/2001/XMLSchema#string"
       ></idFD>
       <hasDim>
        <OLAPDimension rdf:ID="OD_OF1">
         <hasSetDim rdf:resource="#B6"/>
        </OLAPDimension>
       </hasDim>
      </OLAPFact>
     </hasFact>
     <hasFD>
      <FD rdf:ID="FD6">
       <hasBody rdf:resource="#B6"/>
       <idFD rdf:datatype="http://www.w3.org/2001/XMLSchema#string"
       >FD6</idFD>
       <candidateFD rdf:datatype="http://www.w3.org/2001/XMLSchema#boolean"
       >true</candidateFD>
       <hasHead rdf:resource="#H6"/>
      </FD>
     </hasFD>
    </OLAPCube>
    <swrl:Imp rdf:about="http://www.owl-ontologies.com/R3_searchFDCandidate">
     <swrl:body>
      <swrl:AtomList>
       <rdf:rest>
        <swrl:AtomList>
         <rdf:first>
          <swrl:ClassAtom>
           <swrl:classPredicate rdf:resource="#Body"/>
           <swrl:argument1 rdf:resource="#b"/>
          </swrl:ClassAtom>
         </rdf:first>
         <rdf:rest>
          <swrl:AtomList>
           <rdf:rest>
            <swrl:AtomList>
             <rdf:rest>
              <swrl:AtomList>
               <rdf:first>
                <swrl:IndividualPropertyAtom>
                 <swrl:argument2 rdf:resource="#b"/>
                 <swrl:propertyPredicate rdf:resource="#hasBody"/>
                 <swrl:argument1 rdf:resource="#fd"/>
                </swrl:IndividualPropertyAtom>
               </rdf:first>
               <rdf:rest>
                <swrl:AtomList>
                 <rdf:rest>
                  <swrl:AtomList>
                   <rdf:first>
                    <swrl:DatavaluedPropertyAtom>
                     <swrl:propertyPredicate rdf:resource="#candidateDim"/>
                     <swrl:argument1 rdf:resource="#b"/>
```

```
                    <swrl:argument2 rdf:datatype=
                    "http://www.w3.org/2001/XMLSchema#boolean"
                    >true</swrl:argument2>
                   </swrl:DatavaluedPropertyAtom>
                  </rdf:first>
                  <rdf:rest>
                   <swrl:AtomList>
                    <rdf:first>
                     <swrl:DatavaluedPropertyAtom>
                      <swrl:argument2 rdf:datatype=
                      "http://www.w3.org/2001/XMLSchema#boolean"
                      >true</swrl:argument2>
                      <swrl:argument1 rdf:resource="#h"/>
                      <swrl:propertyPredicate rdf:resource="#candidateMes"/>
                     </swrl:DatavaluedPropertyAtom>
                    </rdf:first>
                    <rdf:rest rdf:resource="http://www.w3.org/1999/02/22-rdf-syntax-ns#nil"/>
                   </swrl:AtomList>
                  </rdf:rest>
                 </swrl:AtomList>
                </rdf:rest>
                <rdf:first>
                 <swrl:IndividualPropertyAtom>
                  <swrl:propertyPredicate rdf:resource="#hasHead"/>
                  <swrl:argument2 rdf:resource="#h"/>
                  <swrl:argument1 rdf:resource="#fd"/>
                 </swrl:IndividualPropertyAtom>
                </rdf:first>
               </swrl:AtomList>
              </rdf:rest>
             </swrl:AtomList>
            </rdf:rest>
            <rdf:first>
             <swrl:IndividualPropertyAtom>
              <swrl:propertyPredicate rdf:resource="#depend"/>
              <swrl:argument2 rdf:resource="#b"/>
              <swrl:argument1 rdf:resource="#h"/>
             </swrl:IndividualPropertyAtom>
            </rdf:first>
           </swrl:AtomList>
          </rdf:rest>
          <rdf:first>
           <swrl:ClassAtom>
            <swrl:argument1 rdf:resource="#h"/>
            <swrl:classPredicate rdf:resource="#Head"/>
           </swrl:ClassAtom>
          </rdf:first>
         </swrl:AtomList>
        </rdf:rest>
       </swrl:AtomList>
      </rdf:rest>
      <rdf:first>
       <swrl:ClassAtom>
        <swrl:classPredicate rdf:resource="#FD"/>
        <swrl:argument1 rdf:resource="#fd"/>
       </swrl:ClassAtom>
      </rdf:first>
     </swrl:AtomList>
    </swrl:body>
    <swrl:head>
```

```
    <swrl:AtomList>
     <rdf:first>
      <swrl:DatavaluedPropertyAtom>
       <swrl:argument1 rdf:resource="#fd"/>
       <swrl:propertyPredicate rdf:resource="#candidateFD"/>
       <swrl:argument2 rdf:datatype="http://www.w3.org/2001/XMLSchema#boolean"
       >true</swrl:argument2>
      </swrl:DatavaluedPropertyAtom>
     </rdf:first>
     <rdf:rest rdf:resource="http://www.w3.org/1999/02/22-rdf-syntax-ns#nil"/>
    </swrl:AtomList>
   </swrl:head>
   <swrla:isRuleEnabled rdf:datatype="http://www.w3.org/2001/XMLSchema#boolean"
   >false</swrla:isRuleEnabled>
  </swrl:Imp>
  <FD rdf:ID="FD5">
   <hasHead>
    <Head rdf:ID="H5">
     <candidateMes rdf:datatype="http://www.w3.org/2001/XMLSchema#boolean"
     >true</candidateMes>
     <depend>
      <Body rdf:ID="B5">
       <nbrEltBody rdf:datatype="http://www.w3.org/2001/XMLSchema#int"
       >1</nbrEltBody>
       <idBody rdf:datatype="http://www.w3.org/2001/XMLSchema#string"
       >B5</idBody>
       <bodyHasField rdf:resource="#codeD"/>
      </Body>
     </depend>
     <idHead rdf:datatype="http://www.w3.org/2001/XMLSchema#string"
     >H5</idHead>
     <headHasField>
      <Field rdf:ID="annee">
       <typeField rdf:datatype="http://www.w3.org/2001/XMLSchema#string"
       >int</typeField>
       <idField rdf:datatype="http://www.w3.org/2001/XMLSchema#string"
       >annee</idField>
      </Field>
     </headHasField>
     <headHasField>
      <Field rdf:ID="semestre">
       <idField rdf:datatype="http://www.w3.org/2001/XMLSchema#string"
       >semestre</idField>
       <typeField rdf:datatype="http://www.w3.org/2001/XMLSchema#string"
       >int</typeField>
      </Field>
     </headHasField>
     <headHasField>
      <Field rdf:ID="mois">
       <typeField rdf:datatype="http://www.w3.org/2001/XMLSchema#string"
       >int</typeField>
       <idField rdf:datatype="http://www.w3.org/2001/XMLSchema#string"
       >mois</idField>
      </Field>
     </headHasField>
    </Head>
   </hasHead>
   <idFD rdf:datatype="http://www.w3.org/2001/XMLSchema#string"
   >FD5</idFD>
   <hasBody rdf:resource="#B5"/>
```

```
  </FD>
  <swrl:Variable rdf:ID="type"/>
  <swrl:Imp rdf:ID="searchFieldCandidate">
   <swrla:isRuleEnabled rdf:datatype="http://www.w3.org/2001/XMLSchema#boolean"
   >false</swrla:isRuleEnabled>
   <swrl:body>
    <swrl:AtomList>
     <rdf:first>
      <swrl:ClassAtom>
       <swrl:classPredicate rdf:resource="#Field"/>
       <swrl:argument1 rdf:resource="#f"/>
      </swrl:ClassAtom>
     </rdf:first>
     <rdf:rest>
      <swrl:AtomList>
       <rdf:rest>
        <swrl:AtomList>
         <rdf:rest rdf:resource="http://www.w3.org/1999/02/22-rdf-syntax-ns#nil"/>
         <rdf:first>
          <swrl:BuiltinAtom>
           <swrl:arguments>
            <rdf:List>
             <rdf:first rdf:resource="#typeF"/>
             <rdf:rest>
              <rdf:List>
               <rdf:rest rdf:resource="http://www.w3.org/1999/02/22-rdf-syntax-ns#nil"/>
               <rdf:first rdf:datatype="http://www.w3.org/2001/XMLSchema#string"
               >int</rdf:first>
              </rdf:List>
             </rdf:rest>
            </rdf:List>
           </swrl:arguments>
           <swrl:builtin rdf:resource="http://www.w3.org/2003/11/swrlb#contains"/>
          </swrl:BuiltinAtom>
         </rdf:first>
        </swrl:AtomList>
       </rdf:rest>
       <rdf:first>
        <swrl:DatavaluedPropertyAtom>
         <swrl:propertyPredicate rdf:resource="#typeField"/>
         <swrl:argument2 rdf:resource="#typeF"/>
         <swrl:argument1 rdf:resource="#f"/>
        </swrl:DatavaluedPropertyAtom>
       </rdf:first>
      </swrl:AtomList>
     </rdf:rest>
    </swrl:AtomList>
   </swrl:body>
   <swrl:head>
    <swrl:AtomList>
     <rdf:first>
      <swrl:BuiltinAtom>
       <swrl:builtin                    rdf:resource="http://sqwrl.stanford.edu/ontologies/built-
ins/3.4/sqwrl.owl#select"/>
       <swrl:arguments>
        <rdf:List>
         <rdf:rest rdf:resource="http://www.w3.org/1999/02/22-rdf-syntax-ns#nil"/>
         <rdf:first rdf:resource="#f"/>
        </rdf:List>
       </swrl:arguments>
```

```
        </swrl:BuiltinAtom>
      </rdf:first>
      <rdf:rest rdf:resource="http://www.w3.org/1999/02/22-rdf-syntax-ns#nil"/>
    </swrl:AtomList>
  </swrl:head>
</swrl:Imp>
<FD rdf:ID="FD4">
  <hasBody>
    <Body rdf:ID="B4">
      <nbrEltBody rdf:datatype="http://www.w3.org/2001/XMLSchema#int"
      >1</nbrEltBody>
      <bodyHasField rdf:resource="#gamme"/>
      <idBody rdf:datatype="http://www.w3.org/2001/XMLSchema#string"
      >B4</idBody>
    </Body>
  </hasBody>
  <idFD rdf:datatype="http://www.w3.org/2001/XMLSchema#string"
  >FD4</idFD>
  <hasHead>
    <Head rdf:ID="H4">
      <idHead rdf:datatype="http://www.w3.org/2001/XMLSchema#string"
      >H4</idHead>
      <headHasField>
        <Field rdf:ID="categ">
          <idField rdf:datatype="http://www.w3.org/2001/XMLSchema#string"
          >categ</idField>
          <typeField rdf:datatype="http://www.w3.org/2001/XMLSchema#string"
          >str</typeField>
        </Field>
      </headHasField>
      <depend rdf:resource="#B4"/>
    </Head>
  </hasHead>
</FD>
<swrl:Imp rdf:about="http://www.owl-ontologies.com/R2_searchDimCandidate">
  <swrl:head>
    <swrl:AtomList>
      <rdf:rest rdf:resource="http://www.w3.org/1999/02/22-rdf-syntax-ns#nil"/>
      <rdf:first>
        <swrl:DatavaluedPropertyAtom>
          <swrl:argument2 rdf:datatype="http://www.w3.org/2001/XMLSchema#boolean"
          >true</swrl:argument2>
          <swrl:argument1 rdf:resource="#b"/>
          <swrl:propertyPredicate rdf:resource="#candidateDim"/>
        </swrl:DatavaluedPropertyAtom>
      </rdf:first>
    </swrl:AtomList>
  </swrl:head>
  <swrl:body>
    <swrl:AtomList>
      <rdf:rest>
        <swrl:AtomList>
          <rdf:first>
            <swrl:DatavaluedPropertyAtom>
              <swrl:argument2>
                <swrl:Variable rdf:ID="NbrEltBody"/>
              </swrl:argument2>
              <swrl:propertyPredicate rdf:resource="#nbrEltBody"/>
              <swrl:argument1 rdf:resource="#b"/>
            </swrl:DatavaluedPropertyAtom>
```

```
          </rdf:first>
          <rdf:rest>
            <swrl:AtomList>
              <rdf:rest rdf:resource="http://www.w3.org/1999/02/22-rdf-syntax-ns#nil"/>
              <rdf:first>
                <swrl:BuiltinAtom>
                  <swrl:builtin rdf:resource="http://www.w3.org/2003/11/swrlb#greaterThanOrEqual"/>
                  <swrl:arguments>
                    <rdf:List>
                      <rdf:rest>
                        <rdf:List>
                          <rdf:first rdf:datatype="http://www.w3.org/2001/XMLSchema#long"
                          >3</rdf:first>
                          <rdf:rest rdf:resource="http://www.w3.org/1999/02/22-rdf-syntax-ns#nil"/>
                        </rdf:List>
                      </rdf:rest>
                      <rdf:first rdf:resource="#NbrEltBody"/>
                    </rdf:List>
                  </swrl:arguments>
                </swrl:BuiltinAtom>
              </rdf:first>
            </swrl:AtomList>
          </rdf:rest>
        </swrl:AtomList>
      </rdf:rest>
      <rdf:first>
        <swrl:ClassAtom>
          <swrl:argument1 rdf:resource="#b"/>
          <swrl:classPredicate rdf:resource="#Body"/>
        </swrl:ClassAtom>
      </rdf:first>
    </swrl:AtomList>
  </swrl:body>
  <swrla:isRuleEnabled rdf:datatype="http://www.w3.org/2001/XMLSchema#boolean"
  >false</swrla:isRuleEnabled>
</swrl:Imp>
<StarRelation rdf:ID="SR1">
  <hasFD rdf:resource="#FD6"/>
  <idRel rdf:datatype="http://www.w3.org/2001/XMLSchema#string"
  >R1</idRel>
</StarRelation>
<swrl:Imp rdf:about="http://www.owl-ontologies.com/R4_InferOLAPConcepts">
  <swrla:isRuleEnabled rdf:datatype="http://www.w3.org/2001/XMLSchema#boolean"
  >false</swrla:isRuleEnabled>
  <swrl:head>
    <swrl:AtomList>
      <rdf:rest>
        <swrl:AtomList>
          <rdf:first>
            <swrl:IndividualPropertyAtom>
              <swrl:argument1>
                <swrl:Variable rdf:ID="om"/>
              </swrl:argument1>
              <swrl:argument2 rdf:resource="#h"/>
              <swrl:propertyPredicate rdf:resource="#hasSetMes"/>
            </swrl:IndividualPropertyAtom>
          </rdf:first>
          <rdf:rest>
            <swrl:AtomList>
              <rdf:first>
```

```
            <swrl:IndividualPropertyAtom>
              <swrl:propertyPredicate rdf:resource="#hasDim"/>
              <swrl:argument1>
                <swrl:Variable rdf:ID="of"/>
              </swrl:argument1>
              <swrl:argument2>
                <swrl:Variable rdf:ID="od"/>
              </swrl:argument2>
            </swrl:IndividualPropertyAtom>
          </rdf:first>
          <rdf:rest>
            <swrl:AtomList>
              <rdf:first>
                <swrl:IndividualPropertyAtom>
                  <swrl:argument2 rdf:resource="#om"/>
                  <swrl:argument1 rdf:resource="#of"/>
                  <swrl:propertyPredicate rdf:resource="#hasMes"/>
                </swrl:IndividualPropertyAtom>
              </rdf:first>
              <rdf:rest>
                <swrl:AtomList>
                  <rdf:rest rdf:resource="http://www.w3.org/1999/02/22-rdf-syntax-ns#nil"/>
                  <rdf:first>
                    <swrl:IndividualPropertyAtom>
                      <swrl:propertyPredicate rdf:resource="#hasFact"/>
                      <swrl:argument2 rdf:resource="#of"/>
                      <swrl:argument1>
                        <swrl:Variable rdf:ID="oc"/>
                      </swrl:argument1>
                    </swrl:IndividualPropertyAtom>
                  </rdf:first>
                </swrl:AtomList>
              </rdf:rest>
            </swrl:AtomList>
          </rdf:rest>
        </swrl:AtomList>
      </rdf:rest>
    </swrl:AtomList>
  </rdf:rest>
  <rdf:first>
    <swrl:IndividualPropertyAtom>
      <swrl:argument1 rdf:resource="#od"/>
      <swrl:argument2 rdf:resource="#b"/>
      <swrl:propertyPredicate rdf:resource="#hasSetDim"/>
    </swrl:IndividualPropertyAtom>
  </rdf:first>
 </swrl:AtomList>
</swrl:head>
<swrl:body>
 <swrl:AtomList>
  <rdf:first>
    <swrl:ClassAtom>
      <swrl:argument1 rdf:resource="#fd"/>
      <swrl:classPredicate rdf:resource="#FD"/>
    </swrl:ClassAtom>
  </rdf:first>
  <rdf:rest>
    <swrl:AtomList>
      <rdf:rest>
        <swrl:AtomList>
```

```
<rdf:rest>
 <swrl:AtomList>
  <rdf:first>
   <swrl:IndividualPropertyAtom>
    <swrl:argument1 rdf:resource="#h"/>
    <swrl:propertyPredicate rdf:resource="#depend"/>
    <swrl:argument2 rdf:resource="#b"/>
   </swrl:IndividualPropertyAtom>
  </rdf:first>
  <rdf:rest>
   <swrl:AtomList>
    <rdf:rest>
     <swrl:AtomList>
      <rdf:rest>
       <swrl:AtomList>
        <rdf:first>
         <swrl:DatavaluedPropertyAtom>
          <swrl:argument1 rdf:resource="#b"/>
          <swrl:propertyPredicate rdf:resource="#candidateDim"/>
          <swrl:argument2 rdf:datatype=
          "http://www.w3.org/2001/XMLSchema#boolean"
          >true</swrl:argument2>
         </swrl:DatavaluedPropertyAtom>
        </rdf:first>
        <rdf:rest>
         <swrl:AtomList>
          <rdf:first>
           <swrl:DatavaluedPropertyAtom>
            <swrl:argument1 rdf:resource="#h"/>
            <swrl:argument2 rdf:datatype=
            "http://www.w3.org/2001/XMLSchema#boolean"
            >true</swrl:argument2>
            <swrl:propertyPredicate rdf:resource="#candidateMes"/>
           </swrl:DatavaluedPropertyAtom>
          </rdf:first>
          <rdf:rest>
           <swrl:AtomList>
            <rdf:rest>
             <swrl:AtomList>
              <rdf:first>
               <swrl:ClassAtom>
                <swrl:classPredicate rdf:resource="#OLAPCube"/>
                <swrl:argument1 rdf:resource="#oc"/>
               </swrl:ClassAtom>
              </rdf:first>
              <rdf:rest>
               <swrl:AtomList>
                <rdf:rest>
                 <swrl:AtomList>
                  <rdf:rest>
                   <swrl:AtomList>
                    <rdf:first>
                     <swrl:ClassAtom>
                      <swrl:argument1 rdf:resource="#om"/>
                      <swrl:classPredicate rdf:resource="#OLAPMesure"/>
                     </swrl:ClassAtom>
                    </rdf:first>
                    <rdf:rest          rdf:resource="http://www.w3.org/1999/02/22-rdf-
syntax-ns#nil"/>
                   </swrl:AtomList>
```

```
                    </rdf:rest>
                    <rdf:first>
                      <swrl:ClassAtom>
                        <swrl:classPredicate rdf:resource="#OLAPDimension"/>
                        <swrl:argument1 rdf:resource="#od"/>
                      </swrl:ClassAtom>
                    </rdf:first>
                  </swrl:AtomList>
                </rdf:rest>
                <rdf:first>
                  <swrl:ClassAtom>
                    <swrl:argument1 rdf:resource="#of"/>
                    <swrl:classPredicate rdf:resource="#OLAPFact"/>
                  </swrl:ClassAtom>
                </rdf:first>
              </swrl:AtomList>
            </rdf:rest>
          </swrl:AtomList>
        </rdf:rest>
        <rdf:first>
          <swrl:DatavaluedPropertyAtom>
            <swrl:argument1 rdf:resource="#fd"/>
            <swrl:propertyPredicate rdf:resource="#candidateFD"/>
            <swrl:argument2 rdf:datatype=
            "http://www.w3.org/2001/XMLSchema#boolean"
            >true</swrl:argument2>
          </swrl:DatavaluedPropertyAtom>
        </rdf:first>
      </swrl:AtomList>
    </rdf:rest>
  </swrl:AtomList>
</rdf:rest>
<rdf:first>
  <swrl:IndividualPropertyAtom>
    <swrl:argument1 rdf:resource="#fd"/>
    <swrl:propertyPredicate rdf:resource="#hasHead"/>
    <swrl:argument2 rdf:resource="#h"/>
  </swrl:IndividualPropertyAtom>
</rdf:first>
</swrl:AtomList>
</rdf:rest>
<rdf:first>
  <swrl:IndividualPropertyAtom>
    <swrl:argument2 rdf:resource="#b"/>
    <swrl:propertyPredicate rdf:resource="#hasBody"/>
    <swrl:argument1 rdf:resource="#fd"/>
  </swrl:IndividualPropertyAtom>
</rdf:first>
</swrl:AtomList>
</rdf:rest>
</swrl:AtomList>
</rdf:rest>
<rdf:first>
  <swrl:ClassAtom>
    <swrl:argument1 rdf:resource="#h"/>
    <swrl:classPredicate rdf:resource="#Head"/>
  </swrl:ClassAtom>
</rdf:first>
```

```
            </swrl:AtomList>
          </rdf:rest>
          <rdf:first>
            <swrl:ClassAtom>
              <swrl:argument1 rdf:resource="#b"/>
              <swrl:classPredicate rdf:resource="#Body"/>
            </swrl:ClassAtom>
          </rdf:first>
        </swrl:AtomList>
      </rdf:rest>
    </swrl:AtomList>
  </swrl:body>
 </swrl:Imp>
</rdf:RDF>

<!-- Created with Protege (with OWL Plugin 3.4.4, Build 579)  http://protege.stanford.edu -->
```

RÉFÉRENCES

1. Codd, E.F.; Codd S.B. and Salley C.T, “Providing OLAP to User-Analysts: An IT Mandate”, 1993.

2. LE MOIGNE Jean-Louis, La théorie du systéme general, PUF, Paris 1977, 338p.

3. Inmon et Hackarton, Building the Data Warehouse. John Wiley & Sons, 1996.

4. C. Cauvet, C. Rosenthal-Sabroux (Eds.), Hermes, p. 209-244, 2001.

5. M.C. Wu and A.P. Buchmann, Research Issues in Data Warehousing, BTW, 1997.

6. Devlin, B. A. and Murphy, P. T., “An architecture for a business and information system,” IBM Systems Journal, Volume 27, Number 1, (1988).

7. Ralph Kimball, “The data warehouse toolkit: Practical Techniques for Building Dimensional Data Warehouses, John Wiley and Sons”, ISBN : 0-471-153370, 1996, 2eme ed. : Ralph Kimball, Margaery Ross, “The Data Warehouse Toolkit : The Complete Guide to Dimensional Modeling, 2nd Edition, John Wiley and Sons”, 2002.

8. Marcel P. "Manipulation de Données Multidimensionnelles et Langages de Règles", thèse de doctorat à l’Institut des Sciences Appliquées de Lyon, 1998.

9. Carsten Sapia, Markus Blaschka, Gabriele Hofling, Barbara Dinter,"Extending the E/R Model for the Multidimensional Paradigm", Advances in Database Technologies, ER ’98 Workshops on Data Warehousing and Data Mining, Mobile Data Access, and Collaborative Work Support and Spatio-Temporal Data Management (ER Workshops), LNCS 1552, Springer, p. 105-116, 1998.

10. Nectaria Tryfona, Frank Busborg, Jens G. Borch Christiansen,"starER : A Conceptual Model for DataWarehouse Design", 2nd ACM Intl.Workshop on Data Warehousing and OLAP (DOLAP), ACM Press, p. 3-8, 1999.

11. F. Sha, G. Gardarin, and L. Némirovski "Managing Semi-Structured Data in an object-Relational DBMS", International Database Conference, Hong-Kong, China, July 1999.

12. Enrico Franconi and Ulrike Sattler (1999). A Data Warehouse Conceptual Data Model for Multidimensional Aggregation. Workshop on Design and Management of Data Warehouses (DMDW'99), Heidelberg, Germany, June 1999.

13. Nectaria Tryfona, Frank Busborg, Jens G. Borch Christiansen,"starER : A Conceptual Model for DataWarehouse Design", 2nd ACM Intl.Workshop on Data Warehousing and OLAP (DOLAP), ACM Press, p. 3-8, 1999.

14. Trujillo and Palomar, “An object oriented approach to multidimensional database conceptual modeling (OOMD)”,1998,Proceedings of the 1st ACM international workshop on Data warehousing and OLAP,Washington, D.C., United States.

15. Prat N., Akoka J.,Comyn-Wattiau I., (2006). A UML-based data warehouse design method. Decision Support Systems 42 (2006) 1449-1473.

16. Lujan-Mora, J. Trujillo and I.-Y. Song, A UML profile for multidimensional modeling in data warehouses, Data and Knowledge Engineering 59(3), 725-769.

17. Abello´ A., Samos J. Saltor F. YAM2 (Yet Another Multidimensional Model) : an extension of UML, in Proceedings of the International Database Engineering and Appli-cations Symposium. IEEE Computer Society, pp. 172-181.

18. Golfarelli M., Maio D., Rizzi S., "Conceptual Design of Data Warehouses from E/R Schemes", In Proceedings of the 31st Hawaii International Conference on System Sciences, Kona (Hawaii, USA), 1998.

19. Cabibbo L., Torlone R. A logical Approach to Multidimensional Databases. EDBT’98 Espagne.

20. Moody D., Kortnik M. From Enterprise Models to Dimensional Models : A Methodology for Data Warehouse and Data Mart Design. DMDW’00 Suede.

21. Husemann B., Lechtenbörger J., Vossen G. Conceptual Data Warehouse Design. Proc. Of the Int’l Workshop on Design and Management of Data Warehouses Stockholm Suede, pp 6.1-6.11.

22. A. Tsois, N. Karayiannidis, and T. Sellis. MAC: Conceptual data modelling for OLAP. In Proc. of the International Workshop on Design and Management of Warehouses (DMDW-2001), pages 5–1–5–13, 2001.

23. P. Vassiliadis, A. Simitsis, P. Georgantas, M. Terrovitis, S. Skiadopoulos. A generic and customizable framework for the design of ETL scenarios. Information Systems, vol. 30, no. 7, pp. 492-525, November 2005, Elsevier Science Ltd.

24. Tryfona, Frank Busborg, Jens G. Borch Christiansen, "starER : A Conceptual Model for DataWarehouse Design", 2nd ACM Intl.Workshop on Data Warehousing and OLAP (DOLAP), ACM Press, p. 3-8, 1999.

25. Michael Boehnlein & Achim Ulbrich-vom Ende (1999). Deriving Initial Data Ware house Structures from the Conceptual Data Models of the Underlying Operational Information Systems. Copyright ACM 1999 1-58113-220-4/99/11.

26. Omar Boussaid, Riadh Ben Messaoud , Réémy Choquet, Stééphane phane Anthoard, Laboratoire ERIC, Université Lyon 2 Campus Porte des Alpes. Conception et construction d'entrepôts en XML: EDA - Versailles, juin 2006.

27. O'Neill, J. L., Plausible Reasoning., ACJ, 1987.

28. Patrick Valduriez. Join Indices. ACM transactions on Database Systems, 12(2) :218 - 246, June 1987.

29. Redbrick. "Star Schemas and STAR join Technology." White Paper, Red Brick Systems, Los Gatos (CA), 1995.

30. Ladjel Bellatreche and Kamel Boukhalfa, La fragmentation dans les entrepôts de données : Une approche basée sur les algorithmes génétiques, Revue des Nouvelles Technologies de l'Information (EDA'2005), Juin, 2005, pp. 141-160.

31. Bodo Hüsemann, Jens Lechtenbörger and Gottfried Vossen, Conceptual Data Warehouse Design, 2000.

32. Moody D., Kortnik M. From Enterprise Models to Dimensional Models: A Methodology for Data Warehouse and Data Mart Design. DMDW'00 Suede.

33. Naveen Prakash, Anjana Gosain: Requirements Driven Data Warehouse Development. CAiSE Short Paper Proceedings 2003.

34. Bonifati A, Milano P, Cattaneo F, Ceri S, Fuggetta A, (2001). Designing Data Marts for Data Warehouse. ACM Transaction on Software Engineering and Methodology ACM vol 10 Octobre, pp. 452-483.

35. Phipps C., Davis K. Automating data warehouse conceptual schema design and evaluation. DMDW'02 Canada.

36. Soussi A., Feki J., Gargouri F., « Approche semi-automatisée de conception de schémas multidimensionnels valides. », Revue des Nouvelles Technologies de l'Information – Entrepôts de Données et l'Analyse en ligne (EDA'05), vol. RNTI-B-1, Cépadues éditions, p. 71-89 juin2005.

37. P. Giorgini, S. Rizzi, M. Garzetti. Goal-Oriented Requirement Analysis for Data Warehouse Design. In Proceedings of the ACM Eighth International Workshop on Data Warehousing and OLAP (DOLAP05), 31 October - 5 November, 2005, Bremen, Germany.

38. Jarke M., Jeusfeld M.A., Quix C., Vassiliadis P., "Architecture and quality in data warehouses", Proceedings of the 10th Conference on Advanced Information Systems Engineering (CAiSE '98), Pisa, Italy, June, 8-12, 1998.

39. Jeusfeld M.A., Quix C., Jarke M., "Design and Analysis of Quality Information for Data Warehouses", Proceedings of the 17th International Conference on Conceptual Modeling - ER'98, Singapore, Nov 16-19, 1998.

40. Vavouras A., Gatziu S., Dittrich K.R., "The SIRIUS Approach for Refreshing Data Warehouses Incrementally", BTW'99, pp 80-96, 1999.

41. Zhou G., Hull R., King R., Franchitti J.C., "Using object matching and materialization to integrate heterogeneous databases", Proceedings of the 3rd International Conference on Cooperative Information Systems - CoopIS'95, pages 4-18, 1995.

42. Zhou G., Hull R., King R., Franchitti J.C., "Data integration and warehousing using H2O", IEEE Bulletin of the Technical Committee on Data Engineering, 18(2), 29-40, 1995.

43. Zhou G., Hull R., King R., "Generating Data Integration Mediators that Use Materialization", Journal of Intelligent Information Systems, Volume 6(2), pages 199- 221, May 1996.

44. Hull R., Zhou G., "A Framework for Supporting Data Integration Using the Materialized and Virtual Approaches", Proc. ACM SIGMOD '96.

45. Chawathe S.S., Garcia-Molina H., Hammer J., Ireland K., Papakonstantinou Y., Ullman J.D., Widom J., "The tsimmis project: Integration of heterogeneous information sources", Proceedings of the 10th Meeting of the Information Processing Society of Japan, Marsh 1994, p. 7-18.

46. Papakonstantinou Y., Garcia-Molina H., Widom J.,"Object Exchange Across Heterogeneous Information Sources", IEEE International Conference on Data Engineering, pp. 251-260, Taipei (Taiwan), March 1995.

47. Garcia-Molina H., Hammer J., Ireland K., Papakonstantinou Y., Ullman J., Widom J., "Integrating and Accessing Heterogeneous Information Sources in TSIMMIS", In Proceedings of the AAAI Symposium on Information Gathering, pp. 61-64, Stanford (California, USA), March 1995.

48. Widom J., "Research problems in data warehousing", Proceedings of the 4th International Conference on Information and Knowledge Management – ACM CIKM'95, November 29-December 2 1995, Baltimore (Maryland, USA).

49. Hammer J., Garcia-Molina H., Widom J., Labio W. J., ZhugeY., "The Stanford Data Warehousing Project", IEEE Data Engineering Bulletin, June 1995.

50. Wiener J. L., Gupta H., Labio W. J., Zhuge Y., Garcia-Molina H.,Widom J., "A System Prototype forWarehouse View Maintenance", In Proceedings of the ACM Workshop on Materialized Views : Techniques and Applications, pp. 26-33, Montreal (Canada), June 7 1996.

51. Daniel.K.Schneider, XML Schéma, 2009.

52. Xml Programming Bible (Wiley, 2004).

53. J. Le Maitre, E. Murisasco et M. Rolbert, « SgmlQL, un langage d'interrogation de documents structurés », Actes des 11^{e} Journées Bases de Données Avancées (BDA 1995), Nancy, août 1995, pp. 431-446.

54. S. Abiteboul, D. Quass, J. McHugh, J. Widom, and J. Wiener.The Lorel Query Language for Semi Structured Data. In International Journal on Digital Libraries, pages 68-88, 1997.

55. W3C, 1999.

56. A.Deutsch, M.Fernundez, D.Florescu, A.Levy and D.Suciu.XML-QL : A Query Language for XML. http://www.w3.org/TR/Note-xml-ql.

57. J. Robie, J. Lapp, and D. Schach. XML Query Language (XQL). In The XSL Working Group, World Wide Web Consortium, 1998.

58. Jonathan Robie, Don Chamberlin, and Daniela Florescu, "Quilt: An XML Query Language" , presented at XML Europe, Paris, June 2000.

59. V. Benzaken, G. Castagna, and A. Frisch. "CDuce: An XML-Centric General-Purpose Language", Proceedings of the ACM International Conference on Functional Programming, 2003.

60. V. Aguilera, S. Cluet, P. Veltri, D. Vodislav, and F.Wattez. Querying xml documents in xyleme. ACM SIGIR workshop, 2000.

61. W3C, 2006.

62. D. Duce G.A. Ringland. Approaches to Knowledge Representation: An introduction. Knowledge-Based and Expert Systems Series. John Wiley, Chichester, 1988.

63. John F. Sowa, Knowledge Representation: Logical, Philosophical, and Computational Foundations, Brooks Cole Publishing Co., Pacific Grove, CA, 2000.

64. Petit Larousse, 1998.

65. Robert Stevens, Carole A. Goble and Sean Bechhofer. Ontology-based Knowledge Representation for Bioinformatics , Department of Computer Science and School of Biological Sciences, University of Manchester September 26, 2000.

66. T. R. Gruber. A translation approach to portable ontologies, Knowledge Acquisition,Vol. 5, No. 2, 1993.

67. T.R. Gruber. Towards Principles for the Design of Ontologies Used for Knowledge Sharing. International Workshop on Formal Ontology, Padova, Italy, 1993.

68. Robert Stevens, Chris Wroe, Phillip Lord, and Carole Goble. Ontologies in bioinformatics. In Stefan Staab and Rudi Studer, editors, Handbook on Ontologies in Information Systems, pages 635-657. Springer, 2003.

69. Robert Stevens, Building an Ontology. July 2001. (http://www.cs.man.ac.uk/~stevensr/onto/node12.html#building)

70. D.M. Jones, T.J.M. Bench-Capon, and P.R.S. Visser. Methodologies for Ontology Development. In Proceedings of 15th IFIP World Computer Congress, London, 1998.

71. Fox, M.S., "The TOVE Project: A Common-sense Model of the Enterprise", the Proceedings of the International Conference on Object Oriented Manufacturing Systems, Calgary Manitoba, pp.176-181, 1992.

72. Corcho O, Fernández-López M, Gómez-Pérez A, López-Cima A. Building legal ontologies with METHONTOLOGY and WebODE. Law and the Semantic Web.Legal Ontologies, Methodologies, Legal Information Retrieval, and Applications.Springer-Verlag, LNAI 3369. March 2005.

73. A. Gomez-Perez. Some Ideas and Examples to Evaluate Ontologies. Technical Report KSL-94-65, Knowledge Systems Laboratory, Stanford, 1994.

74. Topquadrant inc., Ontology languages, (http://www.coolheads.com/egov/opensource/topicmap/s167/img21.html)

75. I.M. Keseler, J. Collado-Vides, S. Gama-Castro, J. Ingraham, S. Paley, I.T. Paulsen, M. Peralta-Gil and P.D. Karp. EcoCyc: A comprehensive database resource for Escherichia coli, Nucleic Acids Research 33:D334-7 2005.

76. R.O. Chen, R. Felciano, and R.B. Altman. RiboWeb: Linking Structural Computations to a KnowledgeBase of Published Experimental Data. In Proceedings of the 5th International Conference on Intelligent Systems for Molecular Biology, pages 84-87. AAAI Press, 1997.

77. F. Baader, D. Calvanese, D. McGuineness, D. Nardi, and P. Patel-Schneider, editors. The Description Logic Handbook. Cambridge University Press, 2003.

78. National Centre for Biotechnology Information (NCBI).(http://www.ncbi.nlm.nih.gov/Entrez/)

79. W3C OWL Web Ontology Language Overview http://www.w3.org/TR/owl-features/

80. Ian Horrocks and Peter Patel-Schneider. Reducing OWL entailment to description logic satisfiability. J. of Web Semantics, 1(4):345-357, 2004.

81. McGuinness, D. L. & van Harmelen, F. (2003). OWL Web ontology language overview. http://www.w3.org/TR/owl-features/, August 2003.

82. Volker Haarslev, Ralf Möller. RACER System Description. Proceedings of International Joint Conference on Automated Reasoning, IJCAR'2001, R. Goré, A. Leitsch, T. Nipkow (Eds.),June 18-23, 2001, Siena, Italy, Springer-Verlag, Berlin,pp.701-705.

83. Volker Haarslev , Ralf Möller, Description of the RACER System and its Applications. Proceedings of the International Workshop on Description Logics (DL-2001), Stanford, USA, 1.-3. August 2001, pp. 132-141.

84. OilEd: Bechhofer et al., 2001; Stevens et al., 2001; http://oiled.man.ac.uk

85. Fast Classification of Terminologies (FaCT) is a DL reasoner, Horrocks, 1999; http://www.cs.man.ac.uk/~horrocks/FaCT/

86. Horrocks, I., Patel-Schneider, P. F., Boley, H. & Tabet, S. (2004). SWRL: a semantic web rule language combining OWL and RuleML. http://www.w3.org/ Submission/2004/SUBM-SWRL-20040521/.

87. Foundations of logic programming (second, extended edition). J. W. Lloyd. Springer series in symbolic computation. Springer-Verlag, New York, 1987.

88. PROTÉGÉ (2007). Protégé ontology modeling tool. http://protege.stanford.edu/.

89. RACER (2007). RacerPro reasoner, Racer Systems GmbH & Co. KG. http://www.racer-systems.com/.

90. www.W3C.org

91. http://protege.cim3.net/cgi-bin/wiki.pl?CoreSQWRL

92. http://clarkparsia.com/pellet

93. Bordawekar. R and C. Lang. Analytical Processing of XML Documents: Opportunities and Challenges. In Proceedings of the SIGMD, pages 27-32, 2005.

94. Rizzi S. Conceptual Modeling Solutions for the Data Warehouse. Data Warehouses and OLAP: Concepts, Architectures and Solutions, edited by R.Wrembel and C. Koncilia (2007).

95. T. Niemi, S. Toivonem, M.Niinimaki, and J. Nummenmaa.Ontologies with Semantic Web/grid in data integration for OLAP. International Journal on Semantic Web and Information Systems, Special Issue on Semantic Web and Data Warehousing, 3(4), 2007.

96. Marko Niinimäki, Tapio Niemi: An ETL Process for OLAP Using RDF/OWL Ontologies. J. Data Semantics 13: 97-119 (2009).

97. O.Romero and A.Abello. Automating multidimensional design from ontologies. In DOLAP ’07: Proceeding of the ACM tenth international workshop on Data warehousing and OLAP, pages 1-8, New York, NY, USA, 2007.ACM.

98. M. Gofarelli, S. Rizzi, and B. Vrdoljak, "DataWarehouse Design from XML Sources," In Proc. The 4th ACM Intl Workshop on Data Warehousing and OLAP (DOLAP01), pp. 40-47, Atlanta, 2001.

99. Dong Yang, Rui Miao, Hongwei Wu, Yiting Zhou, “Product configuration knowledge modeling using ontology web language”, Expert Systems with Applications 36 (2009) 4399–4411, 2009.

100. http://wiki.hip.fi/xml/ontology/olapcore.owl

101. F., Calvanese, D., McGuinness, D., Nardi, D., & Patel-Schneider, (2002). The description logic handbook: Theory, implementation and applications. Cambridge University Press.

102. Horrocks, I., Patel-Schneider, P. F., & van Harmelen, F. (2003). From SHIQ and RDF to OWL: the making of a web ontology language. Journal of Web Semantics, 1(1),2–7.

103. Horrocks, I., Patel-Schneider, P. F., Boley, H. & Tabet, S. (2004). SWRL: a semantic web rule language combining OWL and RuleML. http://www.w3.org/Submission/2004/SUBM-SWRL-20040521/.

104. McGuinness, D. L. & van Harmelen, F. (2003).OWL Web ontology language overview. http://www.w3.org/TR/owl-features/, August 2003.

105. Friedman-Hill, E. (2003). Jess in Action: Rule-Based Systems in Java. Manning.

106. Hamadi Abdelkrim, Oukid Khouas Saliha, Boussaid Omar, “Une ontologie pour l’aide à la conception automatisée d’un entrepôt de données XML”, deuxième conférence doctoriales STIC 2011, Tébassa, Algérie.

107. Codd E.F., A Relational Model of Data for Large Shared Data Banks, Communications ACM, V13, N6, Juin 1970, pp 377-387.

108. Codd E.F., Further Normalization of the Database Relational Model, Data Base Systems, Rustin Ed., pp 1017-1021.

109. Amstrong W. W. Dependency structure of data base relationships. Pages 580-583, 1974. In IFIP Congress.

110. Sintek, M. (2007). OntoViz Tab plug-in for Protege. http://protege.cim3.net/cgi-bin/wiki.pl?OntoViz.

111. http://ruleml.org/

112. O’Connor, M. (2007). Protégé SWRL editor, http://protege.stanford.edu/plugins/owl/swrl/.

113. Charles Forgy, "Rete: A Fast Algorithm for the Many Pattern/Many Object Pattern Match Problem", Artificial Intelligence, 19, pp 17–37, 1982.

114. Rizzi S., Abello A., Lechtenborger J., Trujillo J. Research in Data Warehouse Modeling and Design: Dead or Alive? In IXth ACM International Workshop on Data housing and OLAP (DOLAP 06), Arlington, Virginia, USA, pages 3–10. ACM Press, 2006.