N° Ordre 05/2006 – M/IN

République Algérienne Démocratique et Populaire

Ministère de l'Enseignement Supérieur et de la Recherche Scientifique

Université des Sciences et de la Technologie Houari BOUMEDIENE

FACULTE D'ELECTRONIQUE ET D'INFORMATIQUE

Mémoire présenté pour l'obtention du diplôme de
MAGISTERE EN INFORMATIQUE

Organisme d'accueil : **USTHB**

Spécialité : Bases de Données Avancées et Intelligence Artificielle

Intitulé :

Une approche pour l'intégration dans les entrepôts de données (Data Warehouses)

Encadré et proposé par :

P[r] AHMED-NACER Mohamed.

Etudié par :

M[lle] BOUNSIS Lynda.

Devant le Jury composé de :

M[me] ALIMAZIGHI Zaia	Présidente	Maître de conférences	USTHB
M[me] BOUKALA Malika	Examinateur	Maître de conférences	USTHB
M[r] LARABI Slimane	Examinateur	Maître de conférences	USTHB
M[r] SELMOUNE Nazih	Invité	Chargé de Cours	USTHB
M[r] AHMED-NACER Mohamed	Rapporteur	Professeur	USTHB

Dédicaces

Avec tout mon cœur je dédie ce mémoire à tous les membres de ma famille qui sont très chers à mon cœur : ma mère, mon père, ma sœur Nadia et mon frère Yacine. Je le dédie aussi à tous mes amis et à tous mes collègues. Je les remercie pour leur précieuse aide, leur soutien et encouragements réconfortants.

Lynda.

Remerciements

Je tiens particulièrement à remercier Monsieur AHMED-NACER MOHAMED, Professeur et Enseignant-Chercheur à l'USTHB de m'avoir proposer ce sujet et de m'avoir suivi et soutenu durant cette thèse.

Je tiens également à lui témoigner ma gratitude pour ses critiques constructives, ses précieux conseils et encouragements.

Mes remerciements vont également à M^{me} ALIMAZIGHI, M^{r} LARABI, M^{me} BOUKALA et M^{r} SELMOUNE qui me font l'honneur de juger mon travail.

Je tiens aussi à remercier tous ceux qui m'ont aidé de près ou de loin dans mon travail.

Résumé

Toute entreprise possède actuellement d'importants volumes de données, stockées le plus souvent dans des différents médias (bases de données, documents papiers…) et a besoin d'outils permettant une exploitation efficace et performante de ces données pour l'aider dans ses prises de décision.
Les entrepôts de données (ou Data Warehouses) apportent des solutions à cette problématique. Il permet de stocker des données nécessaires à la prise de décision, il est alimenté et mis à jour via des extractions de données provenant des bases de données pouvant être hétérogènes.
Cette pluralité des sources et l'hétérogénéité des informations qu'elles peuvent contenir exige un modèle unifié qui permet à la fois de décrire les structures de données et de les intégrer au sein de l'entrepôt de données. Plusieurs chercheurs préconisent l'utilisation de XML comme format pivot de description et d'échange de données dans le but d'intégrer différentes sources de données compatibles avec le schéma de l'entrepôt. Car dans la plupart des cas, les structures des données sources et leurs sémantiques peuvent être différentes de la structure et la sémantique du Data Warehouse.
Les avantages du langage XML ont fait de lui le meilleur candidat pour l'intégration de données, car en effet, il est riche, clair, simple, puissant, extensible, préserve la sémantique des données, indépendant de toute plate-forme et de tout logiciel.
L'objectif de notre travail est d'apporter une solution pour régler ce problème d'intégration, notre proposition est une méthode exhaustive, simple et automatique qui permet de traiter tous les cas de conflits générés lors de l'intégration de données externes dans un Data Warehouse.

Mots clés : Entrepôt de données, XML, XML Schémas, Bases de Données XML natives, Intégration, Hétérogénéité.

Abstract

Every firm has nowadays an important amount of information, stored generally in different medias (data bases, files…) and needs tools in order to operate efficiently on these data to facilitate taking decision.
The Data Warehouses bring solutions to his problem. They allow stocking all the necessary data to take decision, they are alimented and updated via data extractions coming from different data bases which can be totally heterogeneous.
This heterogeneity needs a unified model which provides the description of the information contained in the different data bases and their integration into the Data Warehouse. A lot of studies show that XML is the best language to use as a unified model. In fact, XML can represent the semantic and the syntax of different data bases.
XML has also different advantages, it is rich, clear, simple, extensible, preserves the semantic in the files and it is independent from any plat-form or software.
The goal of our approach is to provide a way to solve the problems faced when integrating heterogeneous information into Data Warehouse. Our proposition is an exhaustive, simple and automatic method which handles all conflicts cases generated during the data integration into a Data Warehouse.

Key words: Data Warehouses, XML, XML Schemas, XML Native Data Bases, Integration, Heterogeneity.

Table des matières

Chapitre3: XML et les Bases de Données

Chapitre4 : Problématique et Solution Proposée

Introduction générale

Introduction générale

Toute entreprise possède actuellement d'importants volumes de données, stockées le plus souvent dans des différents médias (bases de données, documents papiers…) et a besoin d'outils permettant une exploitation efficace et performante de ces données pour l'aider dans ses prises de décision.

Les entrepôts de données (ou Data Warehouses) apportent des solutions à cette problématique. Un entrepôt de données se définit comme « une collection de données intégrées, orientées sujet, non volatiles, historisées, résumées et disponibles pour l'interrogation et l'analyse ».

Il permet de stocker des données nécessaires à la prise de décision, il est alimenté et mis à jour via des extractions de données provenant des bases de production de l'entreprise, de la saisie des données quotidiennes, de données fournies par les partenaires tels que les fournisseurs, les clients et les administrations publiques ou de données externes à l'entreprise et utile à la prise de décision (documentation juridique, résultats de consultation du Web…).

Cette pluralité des sources et l'hétérogénéité des informations qu'elles peuvent contenir exige un modèle unifié qui permet à la fois de décrire les structures de données et de les intégrer au sein de l'entrepôt de données. Plusieurs chercheurs préconisent l'utilisation de XML comme format pivot de description et d'échange de données dans le but d'intégrer différentes sources de données compatibles avec le schéma de l'entrepôt. Car dans la plupart des cas, les structures des données sources et leurs sémantiques peuvent être différentes de la structure et la sémantique du Data Warehouse.

Les avantages du langage XML ont fait de lui le meilleur candidat pour l'intégration de données, car en effet, il est riche, clair, simple, puissant, extensible, préserve la sémantique des données, indépendant de toute plate-forme et de tout logiciel. De plus, il est évident que les documents XML doivent répondre à une rigueur de contrôle et de structuration à même d'assurer la sécurité des applications traitant les documents XML: contrôle de typage, d'intégrité, de cohérence…. XML permet de structurer un document et de connaître parfaitement cette structure grâce à d'autres langages appelés langages de validation tels que : les DTD et le langage des schémas XSD [1]. Le langage des schémas XSD (écrit en XML) est très puissant car il permet de décrire les types des éléments et leurs occurrences d'une façon complète et précise.

Objectif du travail

L'intégration des schémas de sources hétérogènes fait apparaître des conflits. Les principaux conflits pouvant survenir entre deux schémas sont des conflits de noms et/ou des conflits de types. L'objectif de notre travail est de fournir une méthode exhaustive qui va traiter tous les cas de conflits possibles entre deux schémas XSD, en traitant pour chaque élément des deux schémas leurs noms et leurs types (quelque soit le type, aussi complexe soit il), la comparaison de deux noms se fait grâce au dictionnaire électronique WordNet, et la comparaison des types des éléments nécessitent la comparaison de leurs fils respectifs. Deux éléments peuvent avoir des noms différents syntaxiquement et pourtant équivalents (synonymes), ou avoir des types différents et pourtant équivalents aussi, la méthode proposée dans cette thèse est assez souple pour gérer ce cas de conflit aussi. Les conflits sont simples à traiter dans le cas où les types sont simples (chaînes de caractères, réels, …), par contre le

traitement est un peu plus complexe quand les types sont complexes, l'analyse nécessite à ce moment là le traitement d'un élément pouvant avoir comme fils une séquence d'éléments, un choix entre plusieurs éléments, l'apparition ou non d'un ou plusieurs fils en utilisant l'élément all, le nombre d'occurrence d'éléments…etc. Pour chacun de ces cas, notre système associe un traitement adéquat.

Plan de la thèse

Notre thèse est organisée en deux grandes parties :

La première partie est consacrée au contexte de recherche que nous avons étudié en détail, elle contient une étude approfondie du concept d'un système d'aide à la décision, des Data Warehouses, du langage XML, du langage des schémas XSD et des bases de données XML, pour cela nous avons organisée cette première partie en trois principaux chapitres :

- Le chapitre 1 introduit la notion de système d'aide à la décision tout en spécifiant les principaux éléments de ce dernier qui sont : les sources de données, les entrepôts de données, les magasins de données et les outils d'analyse. Ce chapitre consacre une partie détaillée aux Data Warehouses tout en insistant sur la conception de ce dernier, son alimentation notamment l'extraction et la collecte d'information, le nettoyage et la transformation des données, leur intégration et leur chargement, et aussi leur rafraîchissement. Ce chapitre donne aussi un aperçu sur les Data Marts (magasins de données). L'intégration de données est détaillée dans ce chapitre car elle constitue une étape importante dans la construction d'un entrepôt de données (elle représente 60 à 90% de la construction), de plus, nous nous sommes particulièrement intéressé à cette étape car elle constitue la problématique à laquelle nous avons proposé une solution. Dans le but d'intégrer au sein d'un Data Warehouse plusieurs données provenant de sources hétérogènes, les chercheurs se sont basés sur un format pivot d'intégration qui est le langage XML.

- Le chapitre 2 décrit les concepts et les notions utilisés dans le langage XML tout en insistant sur ses avantages qui ont fait de lui le meilleur candidat pour être un format pivot d'échange d'information. Il décrit aussi les DTD et leur insuffisance à répondre aux besoins de précision et de fiabilité des documents XML. Face à cette insuffisance, le W3C a développé le langage XML schéma. Dans ce chapitre, nous identifions les concepts des schémas et leur force à pouvoir décrire d'une manière très précise le contenu d'un document XML.

- Dans le chapitre 3, nous introduisons la notion de base de données XML et les méthodes de transformation d'une base de données relationnelle ou orientée objet (respectivement de leur schémas) vers une base de données XML native (respectivement de son schéma XSD), aussi, dans ce chapitre, nous mettons en valeur les avantages des bases de données XML native et tout ce qu'elles ont apporté par rapport aux bases de données relationnelles et orientées objet.

La deuxième partie qui est représentée par le chapitre 4 est consacrée à la solution que nous avons apporté à la problématique d'intégration de données hétérogènes, cette solution est basée principalement sur la comparaison de deux schémas XSD associés respectivement au schéma du Data Warehouse et au schéma des données à intégrer. Cette comparaison a pour

but de traiter tous les cas de conflit qui peuvent se poser entre deux schémas et de donner en sortie un résultat booléen : la source de données est intégrable ou non.

Nous terminons cette thèse par une conclusion générale qui résume l'apport essentiel de ce travail et proposons quelques perspectives possibles.

Chapitre1 : Système d'aide à la décision, Data WareHouse et Data Mart

1.1 Introduction

L'information est la nouvelle ressource des entreprises du XXI^{e} siècle. Jusqu'à une période récente l'information était utilisée à des fins de contrôles ou comptables. Mais avec le temps, les entreprises produisent et manipulent de très importants volumes de données. Ces données sont stockées dans les systèmes opérationnels de l'entreprise au sein de bases de données éparpillées et surtout hétérogènes.

Dans le but de mettre toutes ces données dans un seul espace qui va non seulement fournir un système d'aide à la décision qui va puiser ses informations à partir de ces données mais aussi de pouvoir les intégrer au sein d'un espace centralisé, les entreprises ont recours à des systèmes d'aide à la décision spécifiques, basés sur l'approche des entrepôts de données (Data Warehouse) [2]. Cependant ces systèmes restent difficiles à élaborer et sont souvent réalisés de manière empirique, rendant l'évolution du système décisionnel délicate [3].

1.2. Concepts de base

1.2.1 Définition d'un système d'aide à la décision

Décider, d'après le petit Robert, c'est prendre une résolution et adapter une conclusion définitive pour aller dans une direction [4].

Un système d'aide à la décision est un outil destiné à recueillir, organiser, mettre en forme et diffuser des données de manière à en faire des informations décisionnelles [5].

L'informatique décisionnelle apporte à l'entreprise l'information élaborée et exhaustive l'aidant à maintenir sa compétitivité, à accroître sa part de marché et à fidéliser sa clientèle. Les systèmes décisionnels constituent une aide fournie aux acteurs de l'entreprise, leur permettant d'être pro-actifs sur leurs marchés, c'est à dire de décider et d'anticiper en fonction de l'information disponible.

Les systèmes décisionnels permettent de:

- répondre aux besoins d'aide à la prise de décision,
- compléter les systèmes opérationnels,
- fournir l'accès aux informations à des utilisateurs répartis dans l'entreprise.

La démarche adoptée par ces systèmes consiste à:

- utiliser des données cohérentes et vérifiées,
- appliquer les méthodes de modélisation et d'analyses,
- accéder simplement et intuitivement aux données pertinentes,
- établir des règles de décisions,
- mettre en forme des résultats.

L'information décisionnelle fournit des informations nécessaires aux décideurs (directeur, manager…) pour les aider à prendre la meilleure décision en se basant sur les différents outils

d'analyse tel que le processus de l'analyse multidimensionnelle OLAP (On Line Analytical Processing) qui traite le problème selon différentes dimensions, ou bien en se basant sur le Data Mining (DM) qui permet d'explorer intelligemment et efficacement les données du Data Warehouse afin de découvrir les règles régissant les évolutions prévisibles de l'entreprise [2].

1.2.2 Architecture générale d'un système d'aide à la décision

Les systèmes d'aide à la décision regroupent un ensemble d'informations et d'outils mis à la disposition des décideurs pour supporter de manière efficace la prise de décision.

Les éléments d'un système d'aide à la décision

L'architecture des systèmes d'aide à la décision met en jeu quatre éléments essentiels : les sources de données, l'entrepôt de données (Data Warehouse), les magasins de données (Data Marts) et les outils d'analyse et d'interrogation (**Figure1**).

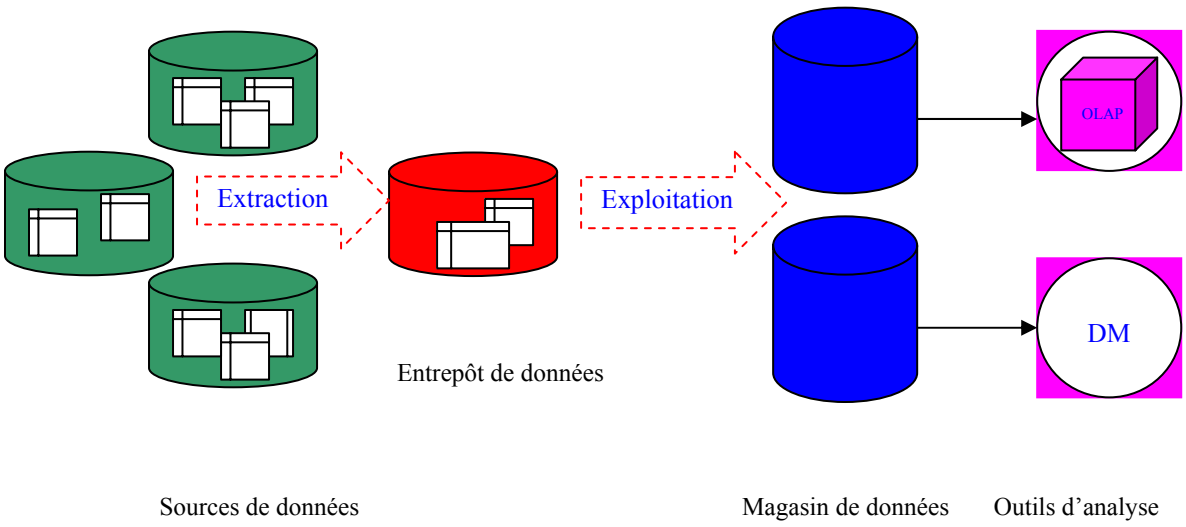


Figure 1: Architecture des systèmes décisionnels

1.2.2.1 Sources de données

Les sources de données sont nombreuses, hétérogènes et autonomes. Elles peuvent être internes (bases de production) ou externes (Internet, bases des partenaires) à l'entreprise et de différents types (relationnelles, orientées objets ou XML natives…) (**Figure2**).

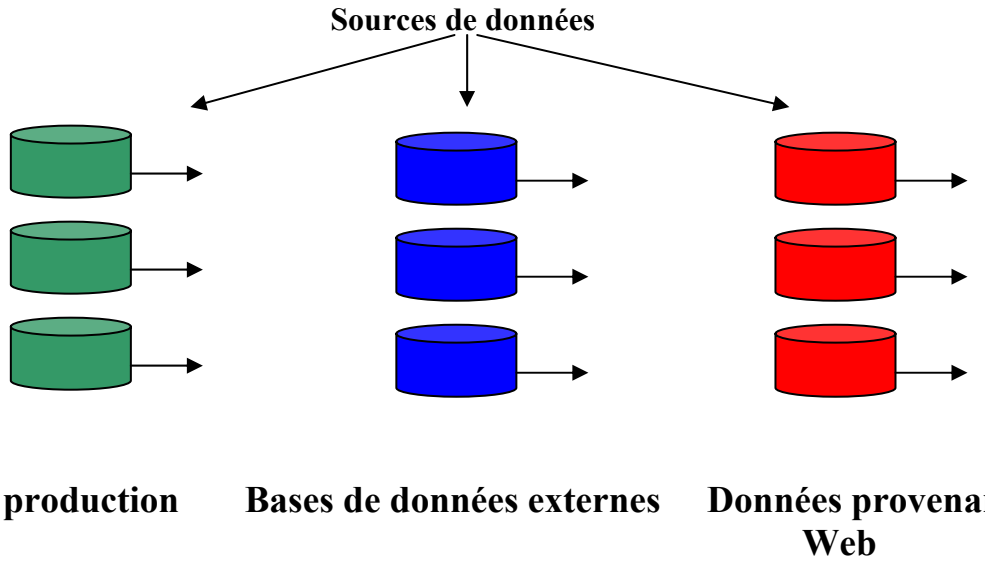


Figure 2: Source de données

1.2.2.2 Entrepôt de données

L'entrepôt de données est le lieu de stockage centralisé des informations utiles pour les décideurs. Il met en commun les données provenant des différentes sources et conserve leurs évolutions (**Figure3**).

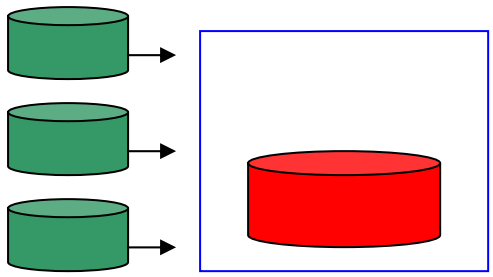


Figure 3: Entrepôt de données.

1.2.2.3 Magasin de données

Les magasins de données sont des extraits, de l'entrepôt, orientés sujet. Les données sont organisées suivant un modèle spécifique pour permettre des analyses rapides à des fins de prise de décision (**Figure4**).

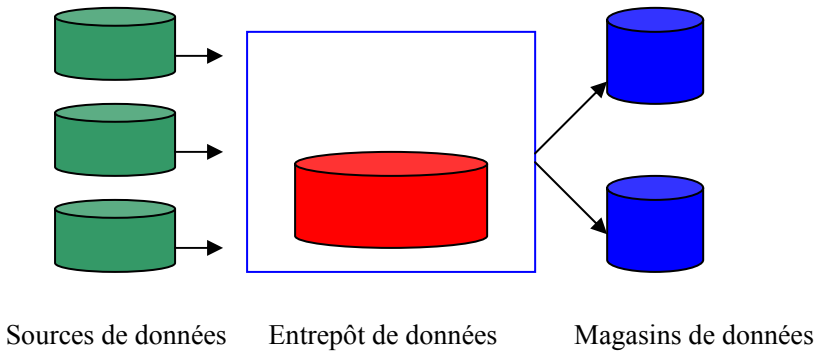


Figure 4: Magasins de données

1.2.2.4 Les outils d'analyse

Les outils d'analyse permettent la manipulation des données stockées dans les magasins de données. Il existe plusieurs outils :

- **L'analyse multidimensionnelle**
  L'analyse multidimensionnelle permet de manipuler les données suivant des axes d'analyse d'informations et de les visualiser au travers des interfaces interactives et fonctionnelles dédiées à des décideurs souvent non informaticiens (décideur, chef de service, manager…).

- **Data Mining**
  Les entreprises et les organisations accumulent des données qui sont souvent sous exploitées alors qu'elles renferment des connaissances stratégiques que même les experts peuvent ignorer.
  Ces réservoirs de données représentent une importante mine d'information dont les entreprises doivent tirer profit. Ils doivent être explorés afin d'en comprendre le sens et d'en découvrir des informations pertinentes et utiles pour des informations de prédiction et de prise de décision. Sont mis en place alors, des mécanismes du Data Mining pour répondre à ce besoin à travers un nouveau domaine de recherche.
  Le Data Mining est un processus d'acquisition progressif de connaissances pertinentes, basé sur la combinaison et l'enchaînement de plusieurs techniques de modélisation, mais c'est l'intelligence de l'exploitant de données qui valorise l'information détenue [4].

1.3 Circuit de l'information dans le système d'aide à la décision

Les données sources ne sont ni sémantiquement cohérentes ni synchrones ni liées entre elles d'une manière adaptée à la perspective décisionnelle. Les environnements d'où viennent ces données se prêtent mal à l'implémentation directe d'applications décisionnelles avancées [6].

La chaîne de mise à disposition des données implique quatre fonctions fondamentales [6] : la fonction de collecte, la fonction d'intégration, la fonction de diffusion et la fonction de présentation :

1. **Fonction de collecte**

   La fonction de collecte est celle qui assure l'approvisionnement du système d'aide à la décision en données primaires puisées dans des sources internes ou externes et pouvant être hétérogènes.

2. **Fonction d'intégration**

   La fonction d'intégration assure la cohérence globale, au moins à l'échelle d'un domaine, des données capturées et leurs mises à disposition en un point unique conformément à un modèle unifié et normalisé.

3. **Fonction de diffusion**

   La fonction de diffusion puise les données dans l'entrepôt central produit et maintenu par la fonction d'intégration, et les met à la disposition des applications sous la forme appropriée à leur analyse.

4. **Fonction de présentation**

   La fonction de présentation gère au moyen des services logiciels plus ou moins élaborés et plus ou moins déterministes, l'accès de l'utilisateur final aux données organisées par la fonction de diffusion.

1.4 Modèles de données

Ces quatre fonctions (fonction de collecte, d'intégration, de diffusion et de présentation) sont organisées techniquement en deux couches (**Figure5**) :

- Système de collecte et d'intégration (**SCI**)
- Système de diffusion et de présentation (**SDP**)

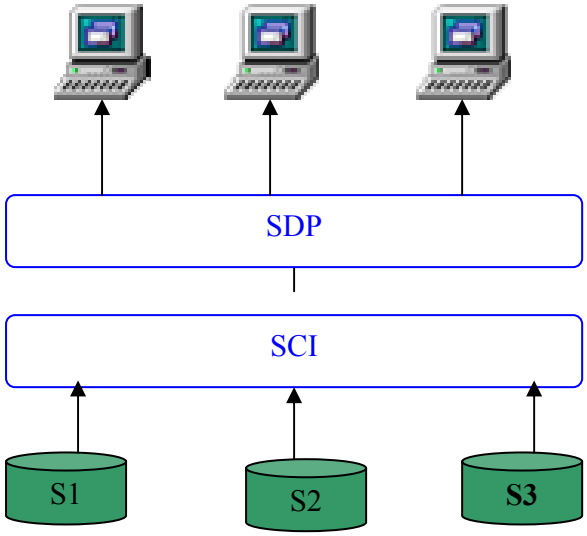


Figure 5: Architecture de référence du système d'aide à la décision

Les systèmes SCI et SDP utilisent trois modèles de données :

1. **Modèle d'intégration**
   La fonction du modèle d'intégration est d'unifier les données opérationnelles, et non de les structurer en contexte d'analyse décisionnel.
   L'élaboration du modèle d'intégration est une œuvre qui prend en entrée des schémas de données hétérogènes et produit en sortie un schéma de données normalisé. La normalisation signifie l'élimination des redondances et l'unification des vocabulaires.

2. **Modèle de diffusion**
   Le modèle de diffusion représente la structure appropriée, selon laquelle les données doivent être mises à la disposition des applications décisionnelles.

3. **Modèle de présentation**
   Le modèle de présentation constitue en quelque sorte le décor. En terme d'architecture, le modèle de présentation n'est qu'un masque qui recouvre, pour l'utilisateur, le modèle de diffusion [6].

1.5 Introduction aux Data Warehouses

Disposer de l'information utile, en avoir plus que ses concurrents, l'avoir préparée à l'avance, la rendre disponible au moment où l'utilisateur en a besoin dans un format compréhensible, sont des objectifs très importants. Toutes les informations d'une entreprise doivent être organisées, coordonnées, intégrées et bien stockées. Le Data Warehouse et ses services rendent ceci possible.
Un Data Warehouse ne s'achève pas, il se construit. Une fois construit, il doit évoluer en fonction des demandes des utilisateurs ou des nouveaux objectifs de l'entreprise, il se situe donc dans une logique d'amélioration imprévisible et fréquente [7], [2], [8].
La difficulté liée à la gestion d'un Data Warehouse principal à fait naître un deuxième espace nommé Data Mart, ciblé sur quelques sujets [53].

1.5.1 Définition d'un Data Warehouse

Le Data Warehouse est une infrastructure permettant de piloter l'entreprise, suivre et mesurer l'impact des décisions stratégiques, il est aussi considéré comme un processus de stockage organisé en de nombreuses données variées pour faciliter l'extraction de l'information importante dans un objectif analytique.

D'après Bill Inmon, un Data Warehouse ou entrepôt de données, est une collection de données, consolidant des informations en provenance des différents systèmes de productions ou opérationnels, ces données sont orientées sujet, intégrées, non volatiles, historisées et organisées pour le support d'un processus d'aide à la décision [7], [9], [49].

Le Data Warehouse doit établir une synergie entre le système d'information et sa stratégie, ce qui implique que le Data Warehouse est le point focal de l'information décisionnelle.

L'information décisionnelle a beaucoup servi à la bonne organisation des données, à une augmentation des performances de l'organisation et à la facilité de prise de décision.

1.5.1.1 Données orientées sujet

Contrairement aux données des organisations traditionnelles généralement organisées par processus fonctionnels, les données de l'entrepôt peuvent être réorganisées autour de thèmes (sujets majeurs de l'entreprise) ce qui facilite leur manipulation.

1.5.1.2 Données intégrées

Toute donnée utile doit être intégrée dans un Data Warehouse quelque soit l'origine d'où elle provient. Mais avant d'être intégrées, les données doivent être mises en forme et unifiées afin d'avoir un état cohérent. Toute donnée doit avoir une description et un codage unique. Dans la réalité, cette phase d'intégration s'avère très complexe, longue, fastidieuse et pose souvent des problèmes de qualification sémantique de données à intégrer. Par expérience, elle représente 60 à 90 % de la charge d'un projet [3], [20].

1.5.1.3 Données historisées

Le Data Warehouse garde trace des données sur une longue période, car l'historisation des différentes valeurs apporte aux analystes la capacité de suivre un indicateur dans le temps et de mesurer les effets de leurs décisions. L'historisation sert de base aux techniques de simulation et de prédiction utilisées par certains outils d'analyse. Il est donc évident qu'un référentiel de temps doit être associé à la donnée afin d'être capable d'identifier une valeur particulière dans le temps.

1.5.1.4 Données non volatiles

C'est une conséquence de l'historisation. Les données de l'entrepôt sont essentiellement utilisées en mode de consultation. Elles ne sont pas modifiées par les utilisateurs.

1.5.2 Caractéristiques d'un Data Warehouse

Les caractéristiques d'un Data Warehouse sont [4], [10] :

- la caractéristique principale d'un Data Warehouse, du point de vue de l'exploitant des données est que ces données soient '' prêtes à l'emploi '', le plus détaillées possible, qualifiées et contrôlées, clairement définies et compréhensibles,

- toutes les informations sont centralisées, quelle que soit la plate-forme ou le système utilisé. Ceci implique un accès à l'ensemble des informations de l'entreprise où qu'elles se trouvent, de telle sorte que les décisions stratégiques soient prises à partir des mêmes données,
- un des points essentiels du Data Warehouse est celui de la qualité du contenu des données,
- le Data Warehouse intègre des données de production avec des données externes et gère des historiques,
- le Data Warehouse regroupe des données de qualité (cohérentes, documentées),
- les décisions ne sont bonnes que lorsque les données sont actualisées. Ce qui permet ainsi de procéder à des réactualisations quotidiennes, voire toutes les heures, de sorte que nous pouvons prendre des décisions en fonction des toutes dernières informations.

1.5.3 Conception d'un Data Warehouse

Il n'y pas une structure figée pour la construction d'un Data Warehouse, tout dépend des objectifs de l'entreprise, pour cela il faut s'approcher de sa stratégie [7], [40].

L'objectif du Data Warehouse se définit en terme métier, il faut donc impliquer les utilisateurs ayant le plus de connaissances dans leur entreprise ou dans leur métier.

Dès la phase de conception il faut administrer les données, c'est à dire mettre en place un référentiel qui décrit, stocke et diffuse les Meta Données.

a) Référentiel du Data Warehouse

Le référentiel du Data Warehouse, c'est l'ensemble des outils nécessaires à la mise en œuvre de la fonction d'administration des données. Il a comme objectifs [7] :

- Assurer la cohérence du système :
  - Respecter la cohérence et la fiabilité des informations,
  - Unifier la représentation des données,
  - Respecter la cohérence des concepts,
  - Vérifier la non-redondance des informations.

- Simplifier techniquement les systèmes d'informations :
  - Diminuer le nombre de fichiers,
  - Unifier la saisie et le stockage des informations,
  - Organiser les mises à jour et la diffusion des informations.

Un référentiel de données pour le Data Warehouse est un référentiel de données dans lequel sont décrits l'organisation et la localisation des données, ainsi que les règles de consolidation des données agrégées. Il est conçu de manière à collecter l'ensemble des données nécessaires à la construction et à l'exploration du Data Warehouse.

La construction du référentiel est un projet qu'il est nécessaire de prévoir en parallèle à celui du Data Warehouse.

b) Les Méta données

Tout système d'information utilise non seulement des données, mais aussi des données sur les données. Ces données représentent toutes les informations nécessaires à l'accès, à la compréhension et à l'exploitation des données du Data Warehouse. Ces dernières désignées par le terme générique de Méta données sont définies dans le **tableau 1** [7] :

Type d'information	Signification
Sémantique	Que signifie la donnée.
Origine	D'où vient-elle, par qui est-elle créée ou mise à jour.
Règle de calcul	Règle de calcul, de gestion.
Règle d'agrégation	Périmètre de consolidation.
Stockage, format	Où, comment est-elle stockée, sous quel format.
Utilisation	Programmes informatiques qui l'utilisent, machines : comment et sur lesquelles, temps de conservation.

Tableau 1: Définition d'une méta donnée

La donnée est liée à d'autres objets du système d'information. Il est donc nécessaire de représenter, écrire et de stocker les informations avec d'autres données. Le **tableau 2** décrit les interactions des données [7], [33].

Type de lien	Signification
Données, sujet	Chaque donnée va être indexée par sujet ou domaine
Structure organisationnelle, structure géographique	Une donnée peut avoir des sens légèrement différents selon la personne qui la manipule
Applications, programmes	Données manipulées par plusieurs applications ou programmes
Tables, colonnes	Données situées dans une ou plusieurs colonnes, tables et bases de données

Tableau 2: Interaction des données

1.5.4 Alimentation du Data Warehouse

La phase de préparation de données est le théâtre d'activités simples de tri et de traitement séquentiel. C'est une phase constituée d'un ensemble de processus qui nettoient, transforment, combinent, archivent et suppriment les doublons. Cette phase prépare les données sources en vue de leur intégration puis de leur exploitation au sein du Data Warehouse (**Figure6**).

Le système de collecte et d'intégration (alimentation d'un Data Warehouse) est le sous-ensemble le plus complexe d'un système d'aide à la décision. C'est aussi le socle sur lequel repose tous les systèmes [44].

La base d'intégration n'est autre, en effet, que l'entrepôt de données proprement dit. Le système de collecte et d'intégration doit être conçue en vue d'un impact minimal sur les applications de production, il ne doit pas perturber l'activité opérationnelle. Mais en outre, tout en respectant cette contrainte, le système de collecte et d'intégration doit assurer une périodicité de rafraîchissement de données en rapport avec les exigences des applications du système de diffusion et de présentation [3], [2], [6], [17].

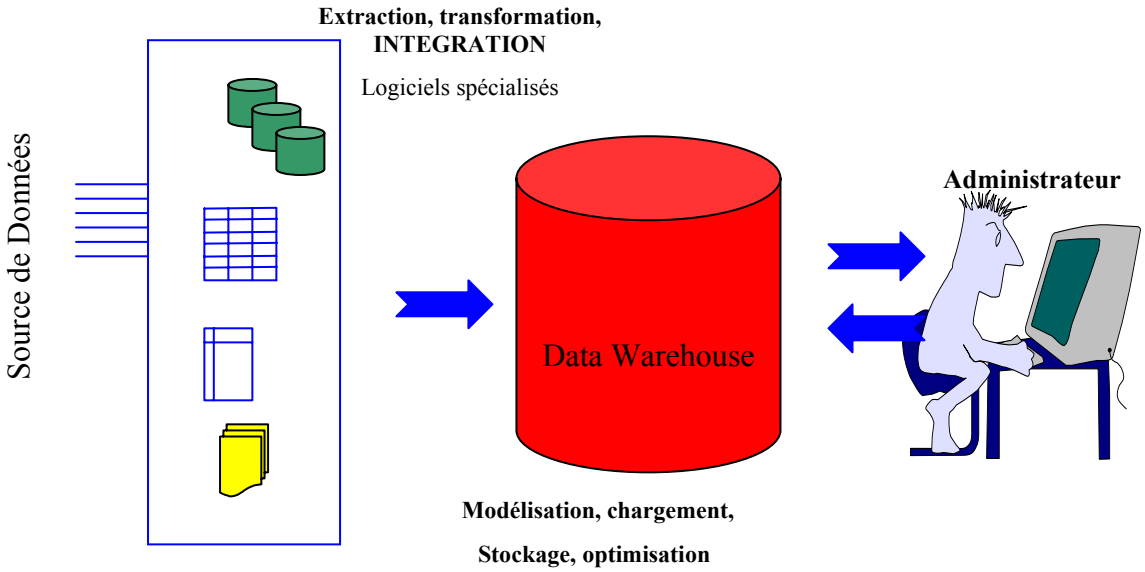


Figure 6: Les besoins et les outils du data Warehouse

1.5.4.1 Extraction

Cette étape consiste à regrouper des données éparpillées, prétraitées dans une mémoire intermédiaire unique avant qu'elles soient utilisées par le client. Cette mémoire peut être une base de données contenant des données intégrées, thématiques, non volatiles et historisées [18], [41].

1.5.4.2 Nettoyage et transformation

L'alimentation d'un Data Warehouse ne se réduit pas à une activité de copie de données d'un environnement à un autre, un problème majeur est d'assurer la cohérence des informations issues de différentes sources.

Pour offrir un support solide et exploitable, le modèle d'intégration doit être normalisé sans compromis, ce qui permet la résolution d'un certain nombre de problèmes relatifs aux sources de données. Parmi ces problèmes nous citons :

- Des noms qui diffèrent d'une source à une autre, peuvent désigner des objets identiques ou semblables,
- un même nom peut, selon la source, désigner des entités ou des propriétés distinctes ayant ou non des propriétés en commun.

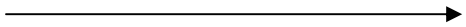
La structure du modèle d'intégration n'est pas une copie conforme de celle de ses sources. Elle s'en différencie à plusieurs points de vues:

1. Une entité du modèle d'intégration peut intégrer des propriétés qui dans les environnements sources appartiennent à des entités différentes.
2. Les entités synthétiques, n'ayant aucune existence dans les sources de données peuvent apparaître dans le modèle d'intégration.

Il existe plusieurs approches de nettoyages [8], [43]**,** parmi elles **:** les outils qui permettent la découverte des rapports entre les données et l'extraction des connaissances de ces données.

Exemple :

ville nous pouvons tirer l'information code ville



1.5.4.3 Les clés

La stabilité des clés primaires des tables est essentielle. Il est fortement recommandé de créer et d'utiliser des clés de remplacement pour les clés primaires de toutes les tables. Les clés de remplacement sont des clés qui sont conservées dans le Data Warehouse en lieu et place des clés provenant des systèmes de données source.

Plusieurs raisons justifient l'utilisation de clés de remplacement (substitution) [5]:

- Dans des systèmes de sources hétérogènes, les tables de données peuvent utiliser des clés différentes pour la même entité :

  Les anciens systèmes qui fournissent des données historiques peuvent respecter un

  système de numérotation différent de celui adopté par un système actuel de traitement

  transactionnel en ligne. Une clé de remplacement identifie de façon univoque chaque

entité de la table du Data Warehouse indépendamment de sa clé source. Dans une entreprise où chaque service utilise son propre système, développé indépendamment de celui des autres services, il est possible que l'un de ces systèmes n'utilise pas les mêmes clés que les autres ou que ses clés soient en conflit avec les données d'un autre service. Une telle solution peut très bien fonctionner sans problème si chaque service génère ses propres rapports de synthèse de façon indépendante, mais elle doit être proscrite dans le cas d'un Data Warehouse, où les données sont consolidées.

- Des modifications dans l'organigramme d'une entreprise peuvent provoquer le déplacement de clés dans la hiérarchie :

  Cette situation est assez courante. Par exemple, si un représentant est muté d'une région à une autre, il pourra être utile pour l'entreprise d'assurer le suivi de deux types d'informations : les ventes réalisées par ce représentant dans son ancienne région d'affectation avant la date de mutation, et celles réalisées dans sa nouvelle région d'affectation, à compter de la date de mutation. Pour représenter cette organisation de données, l'enregistrement du représentant doit figurer à deux emplacements dans la table du Data Warehouse des représentants, ce qui est impossible si le numéro d'identification des employés de l'entreprise fait office de clé primaire dans la table du Data Warehouse. Une clé de remplacement permet au même représentant de figurer à différents emplacements dans la hiérarchie de la table. En l'occurrence, le représentant apparaîtra deux fois dans la table, à l'aide de deux clés de remplacement différentes. Ces clés de remplacement permettent de joindre les enregistrements du représentant aux ensembles de faits correspondant aux différents emplacements de la hiérarchie occupés par l'intéressé.

  Cependant, le numéro d'identification d'un employé doit être stocké dans une colonne distincte de la table, de telle sorte que les informations le concernant puissent être examinées et synthétisées indépendamment du nombre d'occurrences de l'enregistrement de la personne dans la table du Data Warehouse.

1.5.4.4 Intégration et chargement des données:

Avant de pouvoir intégrer les données qui proviennent de différentes sources, nous avons besoin de savoir si ces données correspondent ou non aux informations que contient le Data Warehouse, si c'est données sont cohérente avec la sémantique du Data Warehouse alors nous pouvons les intégrer (elles peuvent même subir une transformation et éventuellement un nettoyage –citées plus haut-) de façon à ce qu'elle soit uniformes et homogènes avec les données incluses dans le Data Warehouse. Mais comment peut-on savoir que des données provenant de sources externes et hétérogènes sont conformes à la sémantique de l'entrepôt ? Encore mieux, existe-il une méthode automatique qui nous permet de savoir qu'une source de données est intégrable et qu'une autre ne l'est pas ?

Plusieurs travaux de recherche ont été effectués autour de ce sujet, et la solution proposée est de construire pour chaque sources de données un modèle qui contient la structure et la sémantique de cette source de données, ainsi chaque source de données aura un modèle qui lui est équivalent, tous les modèles sont écrits dans un même langage, un même format : le format pivot qui permettra une intégration facile, car le schéma des données que contient l'entrepôt de données est représenté aussi sous le même modèle de données. L'avantage d'avoir un format unique de données permettra une intégration facile et résoudra le problème de l'hétérogénéité des données [13], [14].

Pour mettre en œuvre cette solution, les chercheurs ont choisi un langage qui peut jouer le rôle d'un format pivot. Aussi, il faudrait que les différentes sources de données soient facilement transformables en ce langage sans perdre leurs caractéristiques de base et leurs sémantiques. Les avantages du langage XML ont fait de lui le meilleur candidat pour être le format pivot nécessaire à l'intégration de données. En effet, XML est un langage d'échange et de représentation de données, il est riche, claire, simple, puissant, extensible, préserve la sémantique des données, indépendant de toute plate-forme et de tout logiciel. XML permet de structurer un document et de connaître parfaitement cette structure grâce à d'autres langages appelé langages de validation tels que : les DTD et le langage des schémas XSD [1]. Le langage des schémas XSD est très puissant car il permet de décrire les types des éléments d'une façon complète et précise.

De plus la transformation d'une base de données relationnelles ou orientée objets vers un document XML -ou plutôt une base de données XML car nous verrons plus loin qu'un document XML peut être considéré comme étant une base de données- (et vice versa) est très facile et se fait par des logiciels ou des API de manière automatique. Cette transformation (ou plutôt la création d'une vue XML pour un source de données) permet une intégration et un échange facile de données [13], [14], [15], [16].

La création d'une vue XML et d'un schéma lui correspondant pour une source de données se fait grâce à un *Adaptateur* [14], et l'intégration de données dans l'entrepôt (c'est à dire le chargement de données dans le Data Warehouse) se fait grâce à un *Médiateur* [13]. Ce dernier compare le schéma contenu dans l'entrepôt de données avec le schéma des données externes, s'ils sont compatibles alors l'intégration est possible sinon le format de données externes est non-conforme au format du Data Warehouse et donc non-intégrable [1], [37], [48].

Dans les prochains chapitres, nous allons donner en détail la définition du langage XML, des DTD et de leurs limites, le langage des schémas XSD et leur puissance à décrire les types de données, les bases de données XML natives et aussi les mécanismes qui permettent la transformation d'une base de données relationnelle ou orientée objets vers une base de données XML natives [34], [38], [47].

1.5.4.5 Rafraîchissement

Vu la nécessité d'actualisation ainsi que la montée des exigences de réactivité des entreprises, de plus en plus souvent, des informations de la veille au soir sont réclamées. Cette contrainte d'actualité implique une périodicité de rafraîchissement inférieure au mois ou à la semaine [6], [11].

L'opération de rafraîchissement est une fonction répétitive consistant à intégrer périodiquement les nouvelles mises à jour des bases de données sources vers l'entrepôt de données. Ces mises à jour sont détectées par un logiciel appelé *Moniteur de Ressources*.

Ils existent plusieurs méthodes de rafraîchissement [6], [45]:

1) Comparaison du fichier actuel avec une copie de ce fichier établie à une date antérieure. Cette méthode est à la fois fonctionnellement incomplète et matériellement coûteuse. De plus, une donnée créée puis supprimée, dans l'intervalle, sera réputée de ne jamais avoir existé, de même une données modifiée plusieurs fois sera réputée avoir changé qu'une fois.

2) Les SGBD récents permettent de développer des mécanismes capables de provoquer automatiquement le déclenchement d'une procédure choisie quand l'état d'une donnée change. Les déclencheurs (Trigger) sont activés par les changements qui surviennent dans la base de données pour signaler ou enregistrer les changements en temps réel pour le système de collecte et d'intégration.

3) Un procédé plus sophistiqué, basé sur l'exploitation des journaux transactionnels. Ces journaux sont des fichiers spéciaux dans lesquels le gestionnaire de transaction enregistre en séquence tous les événements modifiant l'état des données dont il contrôle. Il contient la trace horodatée de toute création, modification et suppression des données.

Ceci dit, la politique de rafraîchissement est déterminée par l'administrateur de l'entrepôt, selon les besoins des utilisateurs et les caractéristiques des bases de données sources [18], [22], [39].

En résumé, pour faire un KIT d'alimentation d'un Data Warehouse, il nous faut un *Adaptateur, un Médiateur et un Moniteur* qui collaborent afin de transférer des données sources sélectionnées vers l'entrepôt (**Figure7**).

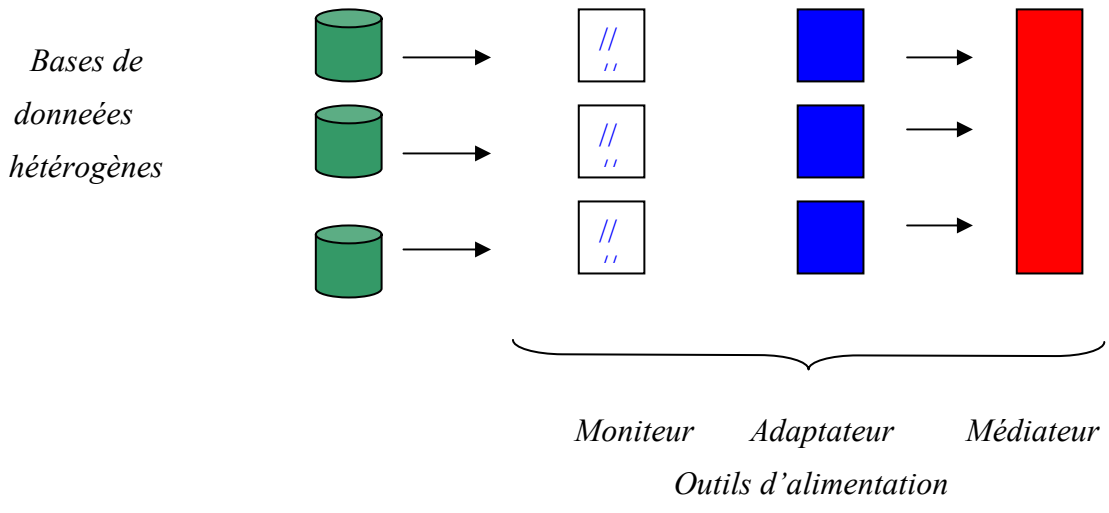


Figure 7: Les composants d'un Kit d'alimentation d'un Data Warehouse

1.6 Data Mart

Le marché de système d'aide à la décision a fortement réagit à la difficulté de gérer un entrepôt de données principal et unique [11]. A côté et souvent en complément se développent des bases de données d'un coût raisonnable ciblées sur quelques sujets limités pour valider rapidement le concept d'information décisionnelle. Ces bases de données sont appelées magasins de données (Data Marts).

1.6.1 Définition d'un Data Mart

Les magasins de données (Data Marts) sont des minis entrepôts de données destinés à quelques utilisateurs d'un département. Ils représentent l'avantage de nécessiter une infrastructure plus légère, de pouvoir être mis en œuvre plus rapidement et d'être mieux centrés sur un problème et sur un sujet [11], [46].

Un Data Mart consiste à extraire une partie de l'information décisionnelle contenue dans l'entrepôt, il s'agit de la partie des données utile pour une classe d'utilisateurs ou pour un besoin d'analyse spécifique, en ce sens ils sont orientés sujet.

1.7 Comparaison entre un Data Warehouse et un Data Mart

La conception d'un système d'aide à la décision doit passer par la séparation de l'entrepôt de données et des magasins de données. L'entrepôt, proprement dit, est le lieu centralisé de toute information pertinente pour les utilisateurs tandis que le magasin de données est un extrait d'entrepôt dédié à un type d'utilisateurs et répondant à un besoin spécifique.

Ainsi, l'entrepôt de données ne supporte pas directement les processus décisionnels tel que OLAP. Cette activité est réservée aux magasins qui améliorent les performances d'interrogations sans se soucier des redondances d'informations; chaque magasin stocke une partie de l'information disponible dans l'entrepôt afin de répondre à un objectif décisionnel précis pour un groupe d'utilisateurs ayant les mêmes besoins. Ces différences de fonctions et d'objectifs se répercutent dans la modélisation de ces deux espaces de stockage :

- L'entrepôt de données vise à stocker l'information décisionnelle disponible dans l'entreprise et à maintenir cette information et ces évolutions au cours du temps.
- Les magasins de données sont dédiés aux analyses décisionnelles tel que OLAP. Les magasins de données sont modélisés selon l'outil d'analyse utilisé. Dans le cas de l'outil d'analyse OLAP, la modélisation multidimensionnelle est la plus appropriée.

Les magasins de données s'adressent aux utilisateurs non experts en informatique et chargés d'effectuer des analyses et de prendre des décisions. Il s'agit généralement d'un petit nombre de personnes qui sont les décideurs de l'entreprise. Ces utilisateurs n'accèdent pas

directement aux informations de l'entrepôt, mais utilisent les magasins de données pour effectuer des interrogations et des analyses au travers d'interfaces visuelles [2], [35].

Cependant, il est aussi utile de pouvoir accéder directement à des données décisionnelles dans l'entreprise ; ceci est utile pour effectuer des analyses ponctuelles ou bien des recherches en vue de la construction de nouveaux magasins. L'accès aux données de l'entrepôt s'adresse à des utilisateurs experts en informatique puisque l'interrogation de l'entrepôt s'effectue souvent au travers de langages de manipulation de type SQL (pour les bases de données relationnelles), XQUERY et XSQL (pour les bases de données XML natives)... [2], [16] (**Figure8**).

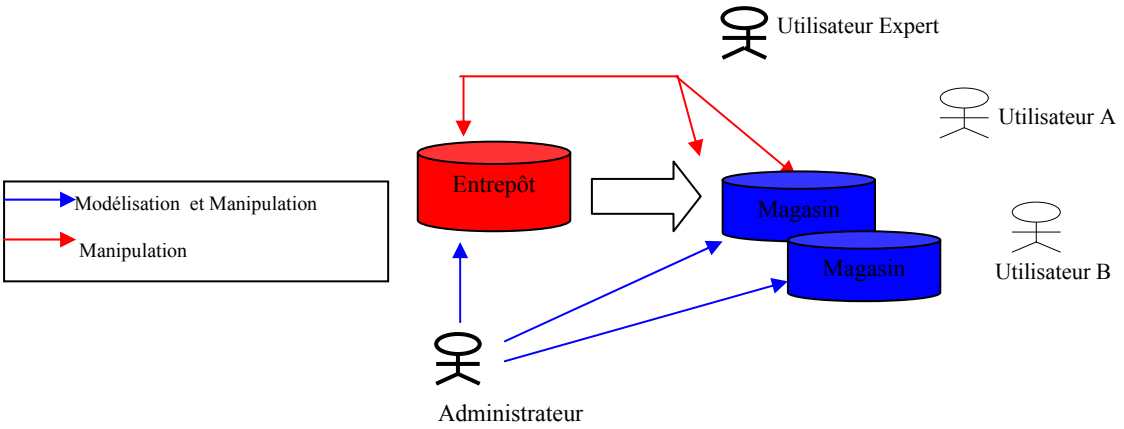


Figure 8: Administration et utilisation des entrepôts et des magasins de données

Pour récapituler, nous allons représenter ces différences dans le **tableau 3** présenté ci-après [7]:

	Data Warehouse	Data Mart
Cible utilisateur	Toute l'entreprise	Département
Implication du service informatique	Elevé	Faible ou moyen
Modèle de données	A l'échelle de l'entreprise	Département
Champ applicatif	Multi sujets	Quelques sujets, spécifique
Source d'alimentation	Sources de production hétérogènes	Data Warehouse

Stockage	Bases de données	Plusieurs bases distribuées
Taille	Centaine de GO et plus	Une à 12 dizaines de GO
Temps de mise en place	9 à 18 mois.	6 à 12 mois (installation en plusieurs étapes)
Coût	Plus de 914 694 Euros	De 76 224 à 457 347 Euros

Tableau 3: Comparaison entre le Data Mart et le Data Warehouse

1.8 Introduction aux bases de données multidimensionnelles

L'expérience a prouvé que le modèle multidimensionnel est le plus adéquat pour l'exploitation des bases de données d'aide à la décision. Ils apportent les meilleurs résultats concernant la facilité d'utilisation et la performance [2]. Pour cela, les magasins de données sont organisés en modèle multidimensionnel dans le cas de l'analyse multidimensionnelle.

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 [2]. La modélisation multidimensionnelle vise à représenter les données sous forme standardisée, intuitive permettant un accès hautement performant.

Le but de cette modélisation est de construire des modèles d'activités décisionnels qui peuvent être compris et parcourus par les utilisateurs finaux.

1.8.1 Architecture d'une base de données multidimensionnelle

Un Data Mart conçu pour effectuer des analyses multidimensionnelles est une base de données multidimensionnelle composée des éléments suivants : les tables dimensions et une ou des table(s) de faits.

1.8.2 Tables de faits

La table de faits est la table principale de tout modèle multidimensionnel, destinée à héberger des données permettant de mesurer l'activité.

Un fait modélise un sujet de l'analyse. Il est formé de mesures correspondant aux informations de l'activité analysée [2]. Ces mesures sont de nature numérique et généralement valorisés de manière continue [3]. Les mesures sont numériques pour permettre de résumer un grand nombre d'enregistrements en quelques enregistrements (nous pouvons les additionner, les dénombrer ou bien calculer le minimum, le maximum ou la moyenne). Elles sont valorisées de façon continue car il est important de ne pas valoriser le fait avec des valeurs nulles [2], [23].

Le choix de grain de fait ou le niveau de détail est un point très important. La granularité d'une table est son niveau de détail le plus fin. Par exemple : le fait de vente : Chaque enregistrement de la tables des faits représente le total des ventes d'un produit dans un magasin par journée.

La table de faits contient une clé multiple composée d'un ensemble de clés primaires. Chaque clé primaire permet de relier la table de faits à une table nommée table dimensionnelle contenant la clé primaire.

1.8.3 Tables dimensionnelles

Une table dimensionnelle ou dimension modélise une perspective de l'analyse. Une dimension se compose de paramètres correspondant aux informations faisant varier les mesures de l'activité [2].

Les dimensions servent à enregistrer les valeurs pour lesquelles sont analysées les mesures de l'activité. Une dimension est généralement formée de paramètres (ou attributs) textuels et discrets sur lesquels portent les clauses de conditions et de groupement au sein de requêtes.

Les paramètres sont discrets, c'est à dire que les valeurs possibles sont bien déterminées et sont des descripteurs constants.

Chaque dimension est définie par sa clé primaire qui assure l'intégrité référentielle avec la table(s) de faits à laquelle(s) elle est reliée [23].

Ces tables dimensionnelles sont équivalentes, en effet, toutes peuvent être vues comme des points d'entrée, symétriquement identiques, dans la table de faits [3].

1.8.4 Modélisation des données

Entre l'océan des données brutes et le foisonnement des demandes d'information, il est évident que c'est impossible d'analyser, de concevoir et de réaliser cette organisation de données sans passer par des modèles. Le rôle pratique d'un modèle consiste à permettre ou à faciliter la pensée et la communication relativement à un objet [21], [36].
L'organisation multidimensionnelle peut être représentée selon trois modèles (schémas):

1.8.4.1 Schéma en étoile

Le schéma en étoile est constitué d'une table de faits et de tables dimensionnelles à un seul niveau de détail.
Chaque dimension est liée à la table de faits à travers sa clé primaire. Les attributs d'une table dimensionnelle peuvent être dénormalisés [12], c'est à dire, l'hiérarchie d'un attribut se trouve dans la même table [19].

1.8.4.2 Schéma en flocon

Le schéma en flocon est un schéma en étoile contenant la normalisation des dimensions, c'est-à-dire chaque dimension peut être représentée par plusieurs niveaux (éclatement d'une dimension en sous tables) en séparant les attributs qui ont un lien fonctionnel entre eux [12]. La modélisation en flocon consiste à décomposer les dimensions du modèle en étoile en sous

hiérarchies. Une hiérarchie organise les attributs d'une dimension selon une relation « est_plus_fin » conformément à leur niveau de détail [19].

1.8.4.3 Le schéma en constellation

La modélisation en constellation est issue du modèle en étoile. Il s'agit de fusionner plusieurs modèles en étoiles qui utilisent des dimensions communes.
Un modèle en constellation comprend donc plusieurs faits et plusieurs dimensions communes [19].

Chapitre2 : Présentation du langage XML, des DTDs et des schémas XSD

2.1 Introduction

XML (abréviation de *eXtensible Markup Language*) est le standard soutenu par le W3C pour le balisage de documents. Il définit une syntaxe générique utilisée pour formater des données avec des balises simples et compréhensibles par l'homme. De ce fait, XML offre une flexibilité d'expression quasi infinie.

XML réunit la simplicité du HTML et la puissance du SGML. Il offre une extensibilité, car il permet à l'utilisateur de définir son propre jeu de balises en préservant ainsi la sémantique des composants d'un document et permet des possibilités de traitement automatisé.

De plus, un document XML se conformant à une structure type explicite (DTD ou schéma XSD) pourra être validé, cela facilite le travail lorsque la structure est complexe.
Le langage XML apporte aussi une indépendance entre la description structurelle des documents et la description de leur réalisation physique qui n'est possible qu'avec une feuille de style (sortie sur écran, imprimante…) ce qui augmente l'espérance de vie d'un document et permet une indépendance par rapport à des logiciels particuliers. Cette indépendance garantit la pérennité à long terme du document et le rend ainsi capable de survivre aux changements technologiques.

Bien que XML soit extrêmement souple, toutes les applications nécessitant de lire des documents XML spécifiques ne peuvent se permettre autant de flexibilité car de nombreux programmes peuvent fonctionner seulement avec certaines applications XML mais pas avec d'autres.
La solution à ces problèmes est la DTD (nous verrons dans le prochain chapitre que les schémas apportent une meilleure solution).
Les DTD sont rédigées dans une syntaxe formelle qui définit avec précision quelles contraintes sont associées à un document XML pour que ce dernier soit conforme à sa DTD.
Le fait, de pouvoir structurer un document et de connaître parfaitement cette structure, permet d'importer des enregistrements d'une base de données sachant que chaque champ correspond tout à fait à la sémantique des composants du document XML.

2.2 Composer un document XML

2.2.1 Définitions

Document bien formé : C'est un document qui obéit à la syntaxe du langage XML, il sera alors déclaré *correct* par un parseur XML.
Document valide : C'est un document *bien formé* qui obéit à certaines contraintes de structuration. La structure ou le format d'un document peut être défini avec les DTD (Document Type Definition) ou bien avec le nouveau langage des Schémas XSD.
Nous distinguons les cas de figures suivants :
Si un document XML ne s'appuie pas sur un schéma, c'est un document bien formé.
Si un schéma est associé à un document et que ce dernier ne respecte pas les contraintes décrites par le schéma alors le document est bien formé mais non valide.
Si un document valide est distribué dans un système d'information (le Web par exemple) sans sa structure (DTD ou Schéma) et sans référence à celle-ci, c'est un document bien formé mais non valide.

Si le schéma est associé à un document et que le document ne viole pas les contraintes du schéma, il est alors bien formé et valide [24].

2.2.2 Structure d'un document

Tout document XML se compose de :

2.2.2.1 Un prologue

Un prologue est facultatif dans un document non valide mais conseillé. Il peut contenir :

La déclaration XML

Elle a la forme suivante :
<?xml version="1.0" encoding='ISO-88591' standalone='yes'?>. Les attributs *version* et *encoding* indiquent respectivement la version du langage XML et le codage des caractères utilisés. La déclaration de la version doit indiquer la conformité du document et automatiser l'identification de la version , tandis que le codage des caractères est obligatoire lorsque le codage par défaut (UTF-8 et UTF-16) n'est pas utilisé car ce dernier convient bien aux utilisateurs anglo-américains. Dans l'exemple cité plus haut, *ISO-8859* est le codage des caractères spécifiques à la langue française. Enfin, l'attribut *standalone* prend la valeur *yes* s'il n'existe pas de déclarations extérieures au document qui doivent être prises en compte sinon la valeur est *no*.

Des instructions de traitement

Leur contenu est transmis à une application en vue de déclencher des traitements. Elles peuvent apparaître dans le corps du document. Ces instructions sont facultatives. Elles ont la forme suivante :**< ?cible arg_1 arg_2…arg_n ?>** où:
c*ible:* est le nom d'une application ou d'une fonction d'application à laquelle l'instruction est destinée.
arg_i *:* est un paramètre (chaîne de caractères) que l'analyseur XML passera inchangé à l 'application identifiée.
Une instruction de traitement permet, par exemple, d'attacher une feuille de style à un document [24].

Déclaration de type de document

Lorsque l'utilisateur veut valider son document par rapport à une DTD, il doit inclure une déclaration de type du document. Elle contient le type du document et certaines déclarations de balisage qui fournissent une grammaire pour une classe de documents. Cette grammaire peut désigner un sous-ensemble externe contenant les déclarations :
< !DOCTYPE nom-type-doc SYSTEM "ressource.dtd" > (où ressource.dtd est le fichier qui contient la structure-type définie). Elle peut aussi contenir ces déclarations directement (sous-ensemble interne) : **< !DOCTYPE nom –type-doc [déclarations]>.** Nous pouvons aussi réunir les deux sous-ensembles comme suit : **< !DOCTYPE nom-type-doc SYSTEM "ressource.dtd" [déclaration]>**.
Une déclaration de balisage est une déclaration de type d'élément, une déclaration de liste d'attributs, une déclaration d'entités ou une déclaration de notation.

Le *nom-type-doc* est le nom de l'élément racine. La déclaration de type de document est indispensable lorsque le document doit être validé [24], [25].

2.2.2.2 L'arbre d'éléments

Un document XML est formé d'une *hiérarchie* d'éléments formant ainsi un *arbre*, chaque élément comporte une balise d'ouverture, un contenu d 'élément et une balise de clôture sauf pour l'élément vide. Il existe un seul élément racine : *le père.* Chaque élément fils est complètement inclus dans son père : il ne peut pas y avoir de recouvrement d'éléments.
Un élément doit être représenté par la forme :
<nom-elt nom-attr$_1$='val$_1$' nom-attr$_2$='val$_2$'…nom-attr$_n$='val$_n$'>contenu de l'élément </nom_elt>.

L'élément vide a la forme suivante :
<nom-elt nom-attr$_1$='val$_1$' nom-attr$_2$='val$_2$' …nom-attr$_n$='val$_n$'/>. Comme il peut être formé d'une balise ouvrante suivi immédiatement de la balise de fermeture :
<nom-elt> </ nom-elt> .
Il est à noter que l'ordre des spécifications des attributs d'un élément n'est pas significatif. Aucun nom d'attribut ne peut apparaître plus d'un fois dans la même balise.
Dans les noms d'élément et attributs, les majuscules et les minuscules ne sont pas équivalentes. La différence Majuscules/Minuscules reste une règle générale en XML.
Les noms d'attributs et éléments sont choisis librement par l'utilisateur, par contre quelques noms d'attributs sont réservés [24].

2.2.2.3 Commentaires

Les commentaires peuvent apparaître dans le contenu d'un élément et dans la déclaration de type du document, toutefois ils ne font pas partie des données textuelles du document et ne sont jamais contenus dans une balise. Un commentaire a la forme suivante : **< !-- Ceci est un commentaire -- >** [24], [26].

2.2.2.4 Le document minimal

Un document minimal est un document *bien formé* contenant un élément vide sans attributs.
Ce document minimal ne contient pas de prologue [24].

2.2.3 Eléments et attributs

2.2.3.1 Balises d'ouverture et de fermeture

Le début de chaque élément XML non vide est marqué d'une balise d'ouverture (balise de début), et sa fin d'une balise de fermeture (balise de fin). La balise d'ouverture commence par «< » et se termine par «> ». Elle contient le nom de l'élément et éventuellement des attributs et leurs valeurs pour décrire certaines propriétés de cet élément. Tandis que la balise de fermeture commence par «</ »et se termine par « > » et contient le nom de l'élément (le même que dans la balise ouvrante). Ce nom spécifie le type de l'élément. Chaque attribut est une paire «nom = 'valeur'» [24], [26].

2.2.3.2 La bonne utilisation des attributs

Il est préférable de faire figurer le contenu informationnel principal d'un document dans le contenu des éléments et de n'utiliser les attributs que pour des paramètres de traitement utilisés par les applications. En utilisant des éléments vides avec des attributs, les information ne seront pas visualisées (nous pouvons toutefois les visualiser avec une feuille de style XSL), cela peut être intéressant dans le cas où on veut *cacher* des informations [24].

2.2.3.3 Indication de langue

Pour faciliter certains traitements linguistiques des documents et leur indexation, l'attribut prédéfini *xml:lang* indique la langue naturelle ou le langage artificiel employé dans le contenu et dans les valeurs d'attributs de tout élément d'un document XML. Sa valeur est un code qui peut être sur deux lettres, défini par la norme ISO 639 (codes pour la représentation des noms de langues) [24], [25].

2.2.3.4 Contenu d'un élément

Un élément, s'il n'est pas vide, peut contenir des données textuelles, des commentaires, des références à des entités, des sections littérales (CDATA) et des instructions de traitement. Il peut aussi contenir aussi d'autres éléments.
L'indexation des documents par les moteurs de recherche les plus répandus se fait sur le contenu des élément, voire sur le contenu de certains éléments bien particuliers et pas sur les valeurs d'attributs.

Sections littérales CDATA : Elles sont employées pour déguiser des blocs de texte contenant des caractères qui seraient autrement identifiés comme du balisage. Leur contenu sera reporté tel qu'il est dans le document source. Une section CDATA pourra alors contenir toutes sortes de caractères sauf la chaîne«]] », et cela en utilisant la forme : **< ![CDATD le document contient toutes sortes de caractères…]]>**.
Pour un document valide, la section CDATA doit être déclarée dans la DTD comme étant du texte (avec #PCDATA).

Données : Pour inclure dans les données les caractères qui ne figurent pas sur le clavier et quelques caractères non autorisés, nous faisons référence, à leurs numéros dans les tables ISO 10646 ou Unicode ou bien à leurs codes prédéfinis [24], [26].

2.3 Structures types de documents (DTD)

2.3.1 Définition

Une DTD (Document Type Definition) est une structure-type prédéfinie. Elle représente l'ensemble de toutes les déclarations contenues -directement ou par référence à des entités externes- dans une déclaration de type de document DOCTYPE. Les déclarations locales au document forment la *partie interne* de la DTD, et celles contenues dans l'entité externe forment sa *partie externe*.

Lorsque le processeur XML lit un document XML, il contrôle en premier lieu les déclarations situées dans le prologue et notamment la déclaration de la DTD. Il va ensuite parcourir le

document XML, en s'arrêtant sur chaque balise. IL compare ensuite la composition effective de l 'élément avec celle définie dans la DTD. Si des différences de structuration apparaissent, le processeur rend comme résultat que le document n'est pas valide car il ne respecte pas les règles de structuration de la DTD.

2.3.2 Entités

Chaque document XML possède une entité appelée *entité document* qui sert de point de départ pour le processeur XML et qui peut contenir le document au complet.
Un document XML peut être constitué d'une ou plusieurs unités de stockage. Ces unités sont appelées *entités*. Chacune à un contenu et toutes, à l'exception de l'entité document et du sous-ensemble externe de la DTD, sont identifiées par un nom d'entité. Les *entités générales* sont destinées à être utilisées dans le contenu du document. Les *entités paramètres,* sont des entités analysables, destinées à être utilisées dans la DTD. Les appels à ces deux entités sont différents et sont reconnus dans des contextes différents. Une entité paramètre et une entité générale de même nom sont deux entités distinctes.
La référence à une entité générale se fait par **&nom-ent ;**. Le remplacement des entités par leur valeur ne se fait que lorsque leur référence est rencontrée dans le contenu d'un élément. Le parseur XML doit rencontrer la déclaration d'une entité avant de rencontrer la référence à cette entité dans le contenu d'un élément. Une entité peut être utilisée dans plusieurs documents. Cela permet une mise à jour et donc une standardisation pérenne [26].

2.3.2.1 Entités prédéfinies

Des caractères propres aux balises peuvent être utilisés dans les données grâce à des codes dits *entités prédéfinies*. Nous y faisons référence par : &lt, &pt, &amp, &apos, &quot pour respectivement : < (lighter) , > (greater) , & (ampersand) , ' (apostrophe) , " (quotation mark).
.

2.3.2.2 Entité interne

Sa *valeur de remplacement* se trouve dans le document lui même. Elle peut aussi faire référence à une entité prédéfinie ou à une entité définie préalablement dans la même déclaration de type de document. L'utilité majeure de l'entité interne est l'abréviation. Elle permet également d'éviter les erreurs de frappe et de modifier facilement, rapidement et complètement le contenu de l'entité (il suffit de modifier une fois la déclaration de l'entité dans la DTD) ainsi le contenu d'une référence peut changer sans que le nom de la référence et son appel changent dans le document XML.

2.3.2.3 Entité externe

Une entité externe permet de segmenter un gros document en plusieurs sous-documents, ou bien représenter une DTD. Sa *valeur de remplacement* fait référence à une valeur se trouvant dans un fichier externe au document lui même. Ces fichiers externes doivent être des documents bien formés.

2.3.3 Entité non XML

Une entité non XML est une entité externe (pouvant être des images, graphiques ou sons). La déclaration de cette entité impose la déclaration de son *format* en lui associant l'application capable de le traiter. Cette déclaration est appelée *notation.*

La référence à une entité non XML est la valeur d'un attribut d'un élément vide et jamais le contenu d'un élément.

2.3.4 Contenu d'une DTD

Une DTD peut contenir des déclarations d'entités générales, des déclarations de notations, des commentaires (vus précédemment), mais aussi :

2.3.4.1 Déclaration d'élément

Sa forme est : **< !ELEMENT nom-type liste >** . *liste* peut être :

- Elément fils :
  - (nom-type-élt-fils$_1$, nom-type-élt-fils$_2$, ...). En utilisant le connecteur de séquence « , » qui permet de déclarer une suite d'éléments *ordonné* (l'ordre est imposé).
  - (nom-type-élt-fils$_1$ | nom-type-élt-fils$_2$ | ...). En utilisant le connecteur de choix « | »qui indique qu'un seul élément fils doit être présent dans le document instance.
- Données : en utilisant le mot clef *#PCDATA* (que du texte).
- Modèle mixte: mélange entre données et éléments fils.
- Contenu libre : *ANY*.
- Elément vide : *EMPTY*.

En plus de ces types d'éléments et des deux connecteurs « , » et « | », des caractères spéciaux (*,+, ?) permettent de spécifier les contraintes d'occurrences des éléments dans les documents instances .
Le caractère « ? » : il suit un élément ou un groupe d'éléments et indique qu'il peut y avoir 0 ou 1 occurrence, il indique le caractère optionnel de l'élément.
Le caractère « * » : il suit un élément ou un groupe d 'éléments et indique qu'il y a 0, 1 ou plusieurs occurrences.
Le caractère « + » : il suit un élément ou un groupe d'éléments et indique qu'il doit y avoir au moins 1 occurrence [24], [25].
Si aucun de ces caractères spéciaux n'apparaît pas, l'élément ou le groupe d'éléments apparaît une et une seule fois.

2.3.4.2 Déclaration de liste d'attributs

Sa forme est : **< !ATTLIST nom-type-élt nom-attr$_1$ type-attr$_1$ occurrence$_1$… nom-attr$_n$ Type-attr$_n$ occurrence$_n$>** .
type-attr$_i$ peut être :

- Attribut *CDATA* : la valeur de l'attribut est une chaîne de caractères prise telle qu'elle est dans le document source.
- Attribut *ENTITY* : la valeur de l'attribut est le nom d'une ou plusieurs entités non XML
- Attribut *NOTATION* : la valeur de l'attribut est le nom d'une notation déclarée précédemment.
- Attribut énuméré : de la forme (par$_1$, par$_2$, ...), la valeur de l'attribut est parmi les valeurs 'par $_i$'.
- Attribut *ID* : Il représente un identifiant pour l'élément, qui doit être unique dans le document.
- Attribut *IDREF* : Il représente une référence à un *ID* se trouvant dans le même document.

- *NMTOKEN* : Il signifie que l'attribut prend comme valeur un nom symbolique quelconque formé de caractères alphanumériques.

Signalons en outre qu'un attribut peut également être une liste de *ID, IDREF, NMTOKEN* ou *ENTITY* séparés par des espaces, grâce aux types *IDS, IDREFS, NMTOKENS* et *ENTITIES* respectivement.

Occurrence$_i$ peut prendre une de ces valeurs:
- ' valeur' ou ' # *DEFAULT* valeur' : la valeur par défaut de l'attribut.
- '*#REQUIRED* ' : la valeur de l'attribut doit impérativement apparaître.
- '*#IMPLIED*' : la présence de l'attribut est facultative dans le document.
- '*#FIXED* valeur' : l'attribut doit prendre la valeur 'valeur' dans le document.

2.3.4.3 Déclaration des entités

L'utilisation des entités paramètres ou générales (à l'exception des entités prédéfinies) impose sa déclaration dans la déclaration du type du document.

- **Entités internes :**
  Son contenu est précisé sous la forme suivante **:< !ENTITY nom-ent 'Valeur de remplacement'>.**

- **Entités externes :**
  Lorsque *nom-ent* est suivi par le mot clef SYSTM la v*aleur de remplacement* est un URL relatif. Sa déclaration se fait cmme suit : **< !ENTITY nom-ent SYSTEM URL>** ou **< !ENTITY nom-ent PUBLIC URL>.**

- **Entités non XML :**
  La déclaration d'une entité non analysable se fait de la même manière que la déclaration d'une entité externe, en rajoutant le mot clé NDATA qui indique que ce fichier est d'un format non XML. Sa forme est :

  < !ENTITY nom-ent SYSTEM URI NDATA format> ou **< !ENTITY nom-ent PUBLIC FPI URI NDATA format> .**

  La notation doit aussi être déclaré et elle a la forme suivante : **< !NOTATION format SYSTEM " application " >.**

- **Entités paramètres :**
  Sachant qu'une *entité générale* est déclarée et référencée dans le même document, une *entité paramètre* est déclarée et référencée dans une même DTD. Donc le texte de remplacement est composé de déclarations de balisage. Sa forme est : **< !ENTITY % nom-ent ' valeur-ent '>** ou **< !ENTITY % nom-ent SYSTEM URL>** ou

< ! ENTITY %nom-ent PUBLIC FPI URL>. La référence à une telle entité se fait par **% nom-ent ;** et ne doit pas apparaître comme valeur d'attribut. [24], [26], [25].

2.3.4.4 Les sections conditionnelles

Les sections conditionnelles offrent une méthode puissante pour créer une DTD permettant la réalisation de versions différenciées destinées par exemple à des catégories différentes de lecteurs. Par versions différenciées, nous ne faisons pas allusions à des différences de mise en pages ou en écran, mais à des différences du contenu même du document. Par exemple, une version de travail d'un rapport contiendra toutes les annotations et commentaires utiles aux auteurs, notamment en cas de production collective, alors qu'une version diffusée plus largement ne contiendra pas ces annotations. Un document peut aussi contenir des sections confidentielles qui ne seront pas incluses dans une version publique mais seulement dans une version interne. Dans tous ces cas, il est particulièrement intéressant de pouvoir générer les différentes versions à partir d'une même DTD. En fait les sections conditionnelles permettent d'ignorer ou d'inclure des entités grâce aux formes :

< !ENTITY nom-ent 'IGNORE'> et **< !ENTITY nom-ent 'INCLUDE'>** [24].

2.3.5 Les avantages et les limites des DTD

L'avènement des DTD a apporté beaucoup d'avantages tels que :

- Il existe un grand nombre d'outils supportant les DTD (tous les outils SGML).
- Les DTD sont répandues car un grand nombre de type de documents sont déjà définis avec les DTD (HTML, XHTML , DocBook , TEI , J2008 , CALS , ...) .
- Une expertise large et de nombreuses années d'expérience sur des applications pratiques.

Mais elles ont quand même quelques limites :

- Les DTD sont écrites dans un langage différent du XML. IL n'est donc pas supporté par les mêmes outils que les documents XML. Ainsi, il est nécessaire d'avoir un autre outil pour traiter les DTD lors de la validation, ceci apporte un surcoût financier et une perte de temps.
- La syntaxe des DTD est très limitées car elle repose sur un type primitif unique qu'est le type texte (*#PCDATA*). Ainsi il est impossible de vérifier le type des éléments du document XML.
- Concernant le nombre d'occurrences des éléments, les caractères spéciaux « * + ? », ne sont pas suffisant. En effet, ils restent très généraux. Il est impossible de définir un nombre d'occurrences plus précisément, ni de définir une plage d'intervalles d'occurrences.

- Les DTD ne supportent pas les espaces de nom de XML, ce qui rend impossible l'import de schémas externes.
- Les fichiers XML peuvent contenir des données qui proviennent ou qui sont destinées à être stockées dans une base de données. Toutefois, outre le fait qu'une base de données est dans la plupart des cas typée, elle est naturellement structurée. Un enregistrement d'une table est généralement composé de plusieurs champs. Or cette notion, tout comme celle d'héritage n'est pas présente dans les DTD. Ainsi il est nécessaire de fournir un gros effort pour établir la correspondance entre le contenu des documents XML et la base de données qui les stocke.

Toutes ces limites des DTD font qu'elles ne peuvent plus correspondre aux besoins d'aujourd'hui. De ce fait, le W3C a développé un autre langage de schémas plus performant qu'est : **XML Schéma** ou le langage des **schémas XSD** [27].

2.4 Introduction aux schémas XSD

Pour palier aux déficiences des DTD, le W3C a conçu un nouveau langage de définition de schémas : **XML Schéma,** connu avec l'extension XSD. Pour cela, il a constitué le XML Schéma Working Group en 1998 afin de développer un langage répondant mieux aux besoins des entreprises concernant leurs documents XML.
Ce groupe de travail a posé les conditions suivantes concernant ce nouveau langage, il doit :

- Etre facilement lisible par un humain et assez simple pour être facilement implémentable.
- Etre écrit en XML.
- Se décrire par lui-même (self-describing).
- Etre utilisable par la majorité des applications qui emploient du XML et permettre l'interopérabilité.
- Etre facilement utilisable sur Internet.
- S'intégrer avec les normes du W3C déjà existantes.

Afin d'atteindre ce but, le projet est devenu « Candidat Recommandation » (c'est à dire au stade terminal) le 24 octobre 2000 et au courant de l'année 2001, il est passé au statut de « Recommandation ».
XML Schéma apporte, en plus des fonctionnalités fournies par les DTD, plusieurs nouveautés :

- En plus du type « texte», XML Schéma intègre les types : booléen, entier, les intervalles de temps … Il est même possible de créer de nouveaux types par ajout de contraintes sur un type existant.
- La notion d'héritage, car les éléments peuvent hériter du contenu et des attributs d'un autre élément. C'est sans doute l'innovation la plus intéressante de XML Schémas.
- Le support des espaces de noms.
- Les indicateurs d'occurrences des éléments sont plus spécifiques, ils permettent d'exprimer toute sorte de contrainte.
- Une grande facilité de conception modulaire de schémas.

XML Schéma va donc permettre à XML d'atteindre son potentiel maximal.

A l'instar des DTD, les schémas permettent de décrire l'imbrication et l'ordre d'apparition des éléments et leurs attributs. Un schéma peut également être vu comme un accord sur un

vocabulaire commun pour des applications particulières concernant les échanges de documents. Un schéma est un modèle pour décrire la structure des informations. Son but est de définir une classe de documents XML.
Ce qui est le plus intéressant dans des schémas XML est que ce sont des documents écrits en XML! Par conséquent, tout schéma XML peut être manipulé par tout éditeur XML.

Il existe plusieurs raisons qui nous poussent à être certains que nos documents sont valides par rapport à des schémas. Exemple : dans le commerce électronique, nous devons être sûrs que l'ordre d'achat reçu via XML est complet et tout ce qui concerne le type de données est correcte (par exemple: les quantités sont positives, les prix sont des décimaux avec deux chiffres après la virgule …etc). Utiliser un schéma et un outil de validation, nous offre une voie standard pour tester nos documents XML.
XML Schéma est composé d'environ 30 éléments et attributs.

Tout document XML Schéma peut débuter par le prologue :
< ? xml version="1.0" encoding="ISO-8859-1" ?> [24].

Cette instruction va être suivie des déclarations d'éléments, d'attributs et de types, ces déclarations appartiennent au contenu de la balise **<schema>** qui est l'élément racine de tout schéma XSD.
Tout élément appartenant aux langage XML Schéma doit être préfixé par le préfixe associé à l'espace de noms des schémas qui est : *http://www.w3.org/2001/XMLSchema*.
Dans tous les exemples qui vont suivre, nous donnons à ce préfixe la valeur *xsd*.

2.5 Déclaration d'éléments

Tout élément est déclaré dans la partie des déclarations d'éléments sous la forme suivante :
<xsd :element name="nom-élt" type="type-élt"/>

L'attribut *name* de l'élément *xsd :element* indique le nom de l'élément.
L'attribut *type* désigne le type de données associé à l'élément déclaré, qui peut être complexe défini par l'utilisateur ou simple prédéfini.
Le type d'élément est complexe si l'élément contient des éléments fils et/ou porte des attributs sinon il est simple.
Cette notion de type est très importante dans XML Schéma [27].

2.6 Déclaration d'attributs

Un attribut est un qualificatif qui s'applique à un élément. A la différence des éléments, les attributs peuvent être uniquement de type simple, car ils ne peuvent avoir ni éléments fils, ni attributs.
La déclaration d'attributs suit la forme suivante :
<xsd :attribute name= "nom-attr" type="type-attr"/>

Les déclarations d'attributs doivent apparaître à la fin des définitions de types complexes puisque ces attributs appartiennent à un élément de type complexe.
Dans la déclaration d'attributs (déclaration locale), nous pouvons utiliser trois attributs optionnels : *use*, *fixed et default.*

Le tableau suivant fournit une description exhaustive des cas selon la valeur des attributs *use*, *fixed* et *default* :

Valeur de l'attribut *use*	Valeur de l'attribut *fixed*	Valeur de l'attribut *default*	Effet
Required	-	-	L'attribut doit apparaître une seule fois, il peur prendre n'importe quelle valeur.
Required	37	-	L'attribut doit apparaître une seule fois, sa valeur doit être 37.
Optional	-	-	L'attribut peut apparaître 0 ou une fois et peut prendre n'importe quelle valeur.
Optional	37	-	L'attribut peut apparaître 0 ou une fois, quand il apparaît, sa valeur doit être égale à 37, s'il est omis, sa valeur est fixée à 37.
Optional	-	37	L'attribut peut apparaître 0 ou une fois, quand il apparaît, sa valeur est libre ; s'il est omis, sa valeur par défaut est 37.
Prohibited	-	-	L'attribut ne doit pas apparaître.

La valeur de l'attribut *fixed* doit être nécessairement conforme au type déclaré de l'attribut.
L'utilisation d'un groupe d'attributs permet d'améliorer la lisibilité et facilite la maintenance d'un schéma. Ainsi un groupe d'attributs qui décrivent un ensemble connexe d'informations peut être définie à un seul endroit : après la définition de types complexes.
Sa syntaxe est :

```
<xsd :attributeGroup   name="nom-du-groupe-d-attributs">
    <xsd:attribute      name="nom-attr1"     type="type-attr1"/>
                              ¦
                              ¦
                              ¦
    <xsd:attribute     name="nom-attrn"    type="type-attrn"/>
</xsd:attributeGroup>
```

Ce groupe peut être référencé dans beaucoup de déclarations et de définitions à l'intérieur de la déclaration de types complexes, de la manière suivante :
<xsd :attributeGroup   ref="nom-du-groupe-d-attributs"/>

Un groupe d'attributs peut contenir un autre groupe d'attributs [27].

2.7 Types de données

XML Schéma fait une distinction importante entre les types de données simples et les types de données complexes.

2.7.1 Types simples

Les éléments de type simple sont des éléments n'ayant pas d'éléments fils ou attributs (un attribut ne peut avoir qu'un type simple). La grande nouveauté de XML Schéma par rapport aux DTD est sa bibliothèque de types intégrée, les listes et les unions.

2.7.1.1 Bibliothèque de types intégrée

XML Schéma comporte une bibliothèque intégrée de types simples (atomiques), elle contient les chaînes de caractères, les entiers, les date etc… ainsi que les types cités dans les DTD : CDATA, NMTOKEN,… repris pour des raisons de compatibilité [27].
Le tableau suivant présente la liste des types définis dans les schémas XML :

Catégorie	Type	Signification
Chaîne de caractères	String	Chaîne
	Name	Nom XML
	Qname	Nom XML qualifié
	Ncname	Nom XML non qualifié
Nombre	Decimal	Décimal
	Boolean	Valeur booléenne
	Float	16 bit floating point number
	Double	32 bit floating point number
	Integer	Infinite accuracy integer
	NonPositiveInteger	Infinite accuracy integer less than 0
	NegativeInteger	Infinite accuracy integer less than 0
	Long	64 bit integer
	Int	32 bit integer
	Short	16 bit integer
	Byte	8 bit integer
	NonNegativeInteger	Infinite accuracy integer more than 0
	PositiveInteger	Infinite accuracy integer more than 1
	UnsignedLong	Unsigned 64 bit integer
	UnsignedInt	Unsigned 32 bit integer
	UnsignedShort	Unsigned 16 bit integer
	UnsignedByte	Unsigned 8 bit integer
Temps	TimeInstant	Date+Time
	TimeDuration	Progress time
	RecuringInstant	
	Date	Date
	Time	Time
XML compatibilité	ID	XML ID
	IDREF	XMLIDREF
	ENTITY	XML ENTITY
	NOTATION	XML NOTATION
	IDREFS	XML IDREFS
	ENTITIES	XML ENTITIES
	NMTOKEN	XML NMTOKEN
	NMTOKENS	XML NMTOKENS
Autres	Binary	Binary
	UriReference	URI
	Language	language

2.7.1.2 Listes

Une suite de types atomiques (simples prédéfinis) séparés par des blancs est un type liste.
De plus, il existe trois types listes intégrés : NMTOKENS, ENTITIES, IDREFS comme nous pouvons en créer d'autres par dérivation de types atomiques. Par exemple NMTOKENS est un type liste constitué d'une série d'éléments NMTOKEN délimités par un espace blanc faisant office de séparateur.

La création de types listes à partir de types complexes ou à partir de types listes n'est pas permise.
La déclaration de listes se fait comme suit :

```
<xsd :simpleType  name="nom-type-liste">
    <xsd:list    itemType="nom-type-simple"/>
</xsd :simpleType>
```

Signalons également que dans cette déclaration *nom-type-simple* peut être un type dérivé.
Il est possible de dériver un type liste à partir du type atomique *string*. Toutefois, un type *string* peut contenir des blancs alors que ces mêmes blancs servent à délimiter les items d'une liste, donc il faut être prudent dans l'utilisation des types listes quand le type de base est *string* (nous verrons la dérivation un peu plus loin) [29].

2.7.1.3 Unions

Un autre type simple est formé de la réunion de plusieurs types simples (type atomiques ou listes) grâce au mot clef *union.*
La déclaration d'une union se fait comme suit :

```
<xsd :simpleType   name="nom-type-union">
    <xsd:union  memberTypes="nom-type1   nom-type2…nom-typen"/>
</xsd :simpleType>
```

$nom\text{-}type_i$ peut être soit un type atomique, un type dérivé ou un type liste déclaré avant.
Plusieurs contraintes, appelés *facettes*, peuvent être appliquées aux listes et unions parce qu'elles sont de types simples (les facettes seront détaillées plus loin) [29].

2.7.2 Types complexes

Un élément qui contient des éléments fils, des attributs ou les deux en même temps, a un type complexe. Les types complexes sont définis pour représenter toutes sortes de modèle de contenu.

Le modèle de contenu (que nous avons appelé dans le langage XML le contenu d'un élément) d'un élément de type complexe nous montre comment ses sous-éléments (éléments fils) et attributs sont susceptibles d'apparaître. Ces sous-éléments et attributs seront décrits à l'intérieur de la balise **<xsd :complexType name="nom-du-type-complexe">,** l'élément complexeType étant le fils de l'élément complexe [30].

2.7.2.1 Le connecteur de séquence

Il permet de définir un type complexe comportant des suites d'éléments. Les sous-éléments seront déclarés à l'intérieur de l'élément **<xsd :sequense>** et seront présents dans le document instance dans le même ordre que dans la déclaration [30].

La syntaxe associée est :

```
<xsd :complexType  name="nom-du-type-complexe">
        <xsd :sequence>
            <xsd :element    name="nom-élt-fils1"    type="type-élt-fils1"/>
                    ¦
                    ¦
            <xsd :element    name="nom-élt-filsn"    type="type-élt-filsn"/>
        </xsd :sequence>
</xsd :complexType>
```

2.7.2.2 Le connecteur de choix

Il permet de choisir entre plusieurs éléments qui seront déclarés à l'intérieur de l'élément **<xsd :choice>** [30].
Un seul élément pourra être présent dans le document instance pour que celle-ci soit valide.

La syntaxe associée est :

```
<xsd :complexType   name="nom-du-type-complexe>
      <xsd :sequence>
            <xsd :element    name="nom-élt-fils"    type="type-élt-fils"/>
            <xsd :choice>
                <xsd :element    name="nom-élt-fils1"    type="type-élt-fils1"/>
                        ¦
                        ¦
                <xsd :element    name="nom-élt-filsn"    type="type-élt-filsn"/>
            </xsd :choice>
            <xsd :element    name="nom-élt-fils"    type="type-élt-fils"/>
        </xsd :sequence>
   </xsd :complexType>
```

2.7.2.3 L'élément all

Les connecteurs de séquence et de choix reproduisent fidèlement les opérateurs des DTD « , » et «|» avec les mêmes propriétés. Alors que l'élément *all* représente un opérateur supplémentaire par rapport aux DTD, il permet à ses éléments fils d'apparaître ou non et dans n'importe quel ordre, seulement, ces éléments doivent être individuels et non pas des groupes d'éléments et apparaissent au plus une fois (c'est à dire les valeurs de *minOccurs* et *maxOccurs* associées sont 0 ou 1) [30].

La syntaxe associée est :

```
<xsd :complexType  name="nom-du-type-complexe">
      <xsd :all>
          <xsd :element          name="nom-élt-fils1"          type="type-élt-fils1"
          minOccurs="val1" maxOccurs="val2"/>
              ¦
              ¦
              ¦
          <xsd :element          name="nom-élt-filsn"          type="type-élt-filsn"
          minOccurs="val1" maxOccurs="val2"/>
       </xsd :all>
</xsd :complexType>
```

2.7.2.4 Les contraintes d'occurrences

Les contraintes d'occurrence sur les éléments sont définies avec les attributs : *minOccurs* et *maxOccurs* où tout nombre entier non négatif peut être utilisé pour fixer leurs valeurs. Pour un nombre illimité d'apparition la valeur spéciale : *unbounded* est donné à *maxOccurs*. La valeur par défaut de ces deux attributs est «1» donc pas nécessaire à spécifier [30].
Lorsqu'il est défini localement, un élément *element* d'un schéma peut porter les attributs

minOccurs et *maxOccurs*, sa syntaxe est :

```
<xsd :element   name="nom-élt"     type="type-élt"     minOccurs="val1"/>
 ou
 <xsd :element   name="nom-élt"     type="type-élt"    maxOccurs="val2"/>
 ou
<xsd :element name="nom-élt"ype="type-élt" minOccurs="val1" maxOccurs="val2"/>
```

Voici un tableau représentant les cas supportés par les DTD traduites par les attributs *minOccurs* et *maxOccurs* :

Contraintes d'occurrences DTD	**Contraintes d'occurrences de XML Schéma**		**Signification**
	Valeur de minOccurs	**Valeur de minOccurs**	
*	0	Unbounded	Indique qu'il y a 0 ou plusieurs occurrences.
+	1(pas nécessaire, valeur par défaut).	Unbounded	Indique qu'il doit y avoir au moins 1 occurrence.
?	0	1(pas nécessaire, valeur par défaut).	Indique qu'il peut y avoir 0 ou 1 occurrence ; il indique le caractère optionnel de l'élément.
rien	1(pas nécessaire, valeur par défaut).	1(pas nécessaire, valeur par défaut).	Indique qu'il y a exactement une et une seule occurrence.

En plus des attributs *minOccurs* et *maxOccurs* un élément déclaré localement peut porter les attributs *fixed* et *default*. Le tableau suivant fournit une description exhaustive des cas selon la valeur des attributs *minOccurs maxOccurs, fixed* et *default :*

Valeur de l'attribut *minOccurs*	**Valeur de l'attribut *maxOccurs***	**Valeur de l'attribut *fixed***	**Valeur de l'attribut *default***	**Effet**
1	1	-	-	L'élément doit apparaître une seule fois, il peut prendre n'importe quelle valeur.
1	1	37	-	L'élément doit apparaître une seule fois, sa valeur doit être 37.
2	unbounded	37	-	L'élément doit apparaître deux fois ou plus, sa valeur doit être 37.
0	1	-	-	L'élément peut apparaître 0 ou une fois et peut prendre n'importe quelle valeur.
0	1	37	-	L'élément peut apparaître 0 ou une fois ; quand il apparaît, sa valeur doit être égale à 37, s'il est omis , le processeur ne lui impose pas une valeur particulière.
0	1	-	37	L'élément peut apparaître 0 ou une fois ; quand il apparaît, sa valeur est libre ; s'il est omis, sa valeur n'est pas imposée ; si cet élément est vide, sa valeur est alors 37.
0	2	-	37	L'élément peut apparaître0, une ou deux fois. La valeur du contenu de cet élément n'est pas fixée mais s'il est vide la valeur considérée par le processeur sera, par défaut, 37.
0	0	-	-	L'élément ne doit pas apparaître.

Grâce à la déclaration des différents types, un processeur XML Schéma validant peut détecter les erreurs de compatibilité de types.
Ainsi, un document XML devient assez fiable pour être enregistré dans une base de données sans aucune forme de contrôle. Cela va simplifier grandement les échanges entre applications.

2.7.2.5 Contenu mixte

XML Schéma fournit la possibilité de construire des schémas dans lesquels les caractères de données (se sont en fait une chaîne de caractères) peuvent apparaître au même niveau que des sous-éléments, nous parlons alors de contenu mixte. La déclaration d'un élément complexe ayant un contenu mixte est la même que la déclaration d'un élément complexe à laquelle nous devons rajouté l'attribut *mixed* positionné à *true* dans la définition du type [30]. Sa syntaxe sera alors :

```
<xsd :complexType   mixed= "true">
        <!- -  déclaration des éléments fils- ->
</xsd :complexType>
```

2.7.2.6 Contenu libre

Un élément peut avoir un contenu libre, il contiendra alors n'importe quelle valeur comme il peut contenir n'importe quelle séquence de caractères, ou un mélange de caractères et de sous-éléments. Pour cela nous utilisons le type *anyType* à l'aide de la syntaxe suivante :
<xsd :element name="nom-élt" type="xsd :anyType"/>
Un contenu d'élément libre est nécessaire dans le cas, par exemple, où nous avons besoin

d'embarquer du balisage étranger dans un élément.

2.7.2.7 Groupe d'éléments

Nous pouvons déclarer un groupe d'éléments grâce à l'instruction :

```
<xsd :group   name="nom-du-groupe">
      <xsd:sequence>
            <!- -  déclaration des éléments fils- ->
      </xsd :sequence>
</xsd :group>
```

et les référencer tous une seule fois sans avoir à les déclarer une deuxième fois et cela à l'aide de la syntaxe suivante :
<xsd :group ref="nom-du-groupe"/> .
L'élément *group* peut avoir comme fils plusieurs éléments *all, choice* ou *sequence* [27].

2.8 Espaces de noms

XML Schéma est une collection de définitions de types et de déclarations d'éléments et d'attributs dont le nom appartient à un espace de nom particulier appelé *espace de nom cible* ou *target namespace*.
Voici quelques exemples pour illustrer les cas où les noms posent problème. Si les deux objets sont du même type, par exemple, supposons que l'on ait défini un type complexe baptisé USSTATES et un type simple du même nom, il y a de facto un conflit. Si les deux objets sont de nature différente, par exemple un élément et un type, disons par exemple que l'on ait défini un type complexe appelé USADDRESS et un élément du même nom, il n'y aura pas de conflit. Si les deux objets sont déclarés à l'intérieur de types différents (c'est à dire qu'il ne s'agit pas d'éléments globaux), par exemple un élément appelé *name* à l'intérieur du type USADDRESS et un autre de même nom à l'intérieur du type ITEM, il n'y a pas de conflit de nom (ces déclarations sont locales).
Si les deux objets sont des types dont l'un est défini par l'utilisateur et l'autre provient de XML Schéma, par exemple un type simple appelé *decimal* défini par l'utilisateur, il n'y a pas de conflit. La raison de l'apparente contradiction du dernier exemple est que les types appartiennent à des espaces de noms différents.
Les espaces de noms permettent de fournir un contexte à un vocabulaire, ce qui facilite la création de schémas et la validation de documents [27], [31].

2.8.1 Espace de noms cible

L'espace de noms cible nous permet de différencier les définitions et déclarations de différents vocabulaires. La valeur de l'attribut *targetNamespace* doit être l'espace de nom associé au document instance. Un espace de noms est déclaré en utilisant un attribut préfixé par *xmlns* (déclaration explicite) ou ayant comme nom *xmlns* (déclaration de l'espace de noms par défaut).
La valeur de l'attribut utilisé pour la déclaration des espaces de noms est une référence URI qui doit être unique et persistante. C'est le nom identifiant l'espace de noms.
Considérons l'exemple :

```
< ?xml  version="1.0"  encoding="ISO-8859-1" ?>
<xsd :schema  xmlns :xsd="http://www.w3.org/2000/10/XMLSchema"
xmlns="http://site.voila.fr/xmlschema/ressources/contacts"
targetNamespace="http://site.voila.fr/xmlschema/ressources/contacts"
<!- - déclarations et définitions- ->
<xsd :element  name="contacts"  type="typeContacts"/>
<xsd :element  name="remarque"  type="xsd:string"/>
</xsd:schema>
```

Dans l'exemple précédent, l'espace de noms associé au document des contacts est l'identificateur de ressource uniforme URI 'http://site.voila.fr/xmlschema/ressources/contacts'.
Par conséquent, tout élément de ce schéma précédé du qualificatif associé à l'espace de noms cible est interprété par le processeur XML comme un élément de cet espace de noms. L'espace de noms cible joue un rôle important dans le processus d'identification.
Le nom de l'élément ne peut alors en aucun cas entrer en conflits avec le nom d'un élément issu d'un autre espace de noms s'il est adéquatement qualifié [27], [31].

2.8.2 Qualification explicite et implicite

Nous avons associé l'espace de nom de XML Schéma avec le préfixe *xsd :* c'est ce qu'on appelle une qualification *explicite*, ce qui est d'usage. N'importe quel autre préfixe aurait pu être choisi. Pour être valide, tout élément appartenant au langage XML schéma doit être précédé du préfixe associé à l'espace de noms des schémas XSD.
Il existe également un procédé de qualification dit *implicite* qui utilise l'espace de noms par défaut.
Tout document XML peut avoir un espace de noms par défaut, c'est à dire que les éléments non préfixés du document appartiennent à l'espace de noms par défaut, ils seront quand même valides [27], [31].

2.8.3 Déclarations locales et globales

Un élément ou un attribut est déclaré *global* lorsqu'il est fils de l'élément *schema.* Par conséquent la déclaration globale d'un élément lui permet d'être au plus haut niveau dans une instance d'un document c'est à dire l'élément racine.
Les éléments ou attributs déclarés à l'intérieur d'un type complexe (par exemple) sont dits *locaux.*
Nous pouvons également référencer un élément défini *globalement* à l'intérieur de la déclaration d'un type complexe, c'est à dire qu'il sera référencé à la suite d'un élément déclaré *localement* avec la précaution d'utiliser le mot clef *ref* .

Sa syntaxe est :
<xsd :element ref="nom-élt-global"/>

Si un élément est utilisé plusieurs fois, mieux vaut le déclarer comme global (déclarer son nom et son type) et le référencer en tant qu'un élément local avec l'attribut *ref* sans avoir à le redéclarer à chaque fois.
La principale contrainte pour une déclaration globale dans un schéma XML est que le nom de l'élément ou de l'attribut doit être unique. Les déclarations globales ne peuvent pas contenir de référence. Elles ne peuvent donc pas contenir l'attribut *ref.* Il y a également une différence entre les éléments ou attributs déclarés *localement* et *globalement* au niveau de la qualification dans les documents instances [27], [31].

2.8.4 Qualification dans les documents instances

Dans le document source, les noms d'élément et attributs sont qualifiés par le préfixe déclaré (tout préfixe, sauf *xml* et *xmlns*, doit être déclaré comme étant un attribut de déclaration d'espace de noms). Notons que contrairement au schéma lui même, le document instance a des attributs qualifiés car aucune balise ne peut contenir deux attributs non qualifiés qui ont des noms identiques ou qui ont les mêmes noms et sont qualifiés avec des préfixes associés au même espace de noms.

La déclaration de l'espace de noms est appliquée à l'élément où elle est spécifiée et à son contenu sauf si un autre espace de noms y est déclaré.
Exemple :

<pref : pére xmlns :pref = URI elementFormDefault =qualified >
<pref : fils$_1$/>
¦
<pref : fils$_n$ />
</pref :père>

Si l'attribut utilisé est *xmlns* alors sa valeur est l'espace de noms par défaut, qui est appliqué à l'élément où il est déclaré si celui-ci n'est pas qualifié et à tous les éléments non préfixés dans son contenu. Dans ce cas, la référence URI peut ne pas y être et alors les éléments non préfixés n'appartiennent à aucun espace de noms. Notons que l'espace de noms par défaut ne s'applique pas aux attributs [27].

2.8.5 Eléments et attributs globaux

Dans le document instance, les éléments et attributs globaux doivent être qualifiés explicitement ou implicitement.
Il n'y a pas que l'élément racine du document instance qui doit être déclaré *globalement.* Pour des raisons que nous verrons plus tard, tout élément peut être déclaré *global* dans le document instance et référencé *localement* [27].

2.8.6 Eléments et attributs locaux

Un auteur de schéma XML peut décider si les éléments et les attributs déclarés localement doivent être qualifiés par un espace de noms, en utilisant une qualification explicite ou implicite. Il dispose pour cela de deux méthodes, l'une spécifiant globalement la qualification

des attributs et éléments locaux et l'autre spécifiant de façon ponctuelle la qualification d'un attribut ou d'un élément.
La première méthode consiste à spécifier globalement si la qualification pour les éléments et attributs locaux est requise dans les documents instances. Il existe deux attributs :*elementFormDefault* et *attributeFormDefault*, portés par l'élément *schema.* Les deux attributs ont pour valeur par défaut *unqualified* ce qui signifie que la qualification n'est pas requise.
L'affectation de l'attribut *elementFormDefault* à *qualified* implique que tout élément déclaré localement doit être précédé du préfixe associé.
La seconde méthode consiste à spécifier pour certains éléments ou attributs locaux si la qualification est requise, à l'aide de l'attribut *form*. Cet attribut peut être porté par l'élément *element* et l'élément *attribute* de XML Schéma. Notons que la valeur de cet attribut (*qualified* ou *unqualified)* écrase la valeur de l'attribut de *elementFormDefault* ou de *attributeFormDefault* selon qu'il est porté par l'élément *element* ou par l'élément *attribute* respectivement.
Les attributs qui doivent être qualifiés doivent être explicitement préfixés parce que la spécification XML-Namespace ne fournit aucun mécanisme pour définir un espace de nom par défaut pour les attributs (c'est le cas du préfixe propre au document instance).
Les attributs qui n'ont pas besoin d'être qualifiés apparaissent dans les instances de documents sans préfixe, ce qui est le cas classique.
Enfin, il est clair que cette qualification d'éléments ou d'attributs semble lourde et compliquée, cependant la qualification implicite avec les espaces de noms par défaut peut être utilisée [27].

2.8.7 Schéma sans espace de noms cible

Les déclarations d'éléments dans un schéma sans espace de noms cible valident uniquement les éléments non qualifiés du document instance. C'est à dire qu'elles valident les éléments pour lesquels il n'y a aucune qualification, ni explicite, ni implicite. Donc, pour valider un document XML 1.0 traditionnel n'utilisant pas d'espace de noms, il faut utiliser un schéma sans espace de nom cible.
De plus, quand un schéma est conçu sans espace de noms cible, il est fortement recommandé que tous les éléments et types en provenance de l'espace de noms de XML Schéma soient *explicitement* qualifiés par un préfixe tel que *xsd*. La raison en est que, si cette association est faite par défaut, les références aux types issus de XML Schéma pourront ne pas être distinguées de celles faisant références aux types définis par l'utilisateur [27].

2.9 Dérivation de types

En plus des types intégrés et la possibilité de créer des types complexes, XML Schéma est doté d'un mécanisme d'*héritage*, il propose deux techniques de dérivation de type : par *extension* du type ancêtre ou par *restriction*. Ces deux méthodes s'appliquent aussi bien aux types simples que complexes. Ce qui les rend très puissants.

2.9.1 Restriction

2.9.1.1 Restriction de types simples (facettes)

Le mécanisme de dérivation par restriction permet de créer de nouveaux types simples (ou atomiques) à partir de la bibliothèque de types intégrée à XML Schéma.

Nous introduisons à cette fin la notion de *facettes*, qui sont en réalité des contraintes applicables sur un type simple particulier de *base*. Une *facette* permet de restreindre l'ensemble des valeurs légales d'un type de base.
Il existe un nombre important de *facettes* qui permettent de :

- Fixer la longueur d'un type simple, restreindre sa longueur maximale et minimale. Par exemple la facette *length* permet de fixer la longueur d'un type chaîne de caractères:

```
<xsd :simpleType  name="nom-de-mon-nouveau-type-simple">
    <xsd :restriction   base="xsd :le-type-simple-prédéfini">
        <xsd :length   value="valeur-entière-positive"/>
    </xsd :restriction>
</xsd :simpleType>
```

- Enumérer toutes les valeurs possibles d'un type. Par exemple la facette *enumeration* peut être utilisée comme suit :

```
<xsd :simpleType  name="nom-de-mon-nouveau-type-simple">
    <xsd :restriction   base="xsd :le-type-simple-prédéfini">
        <xsd :enumeration  value="valeur1"/>
                ¦
                ¦
        <xsd :enumeration  value="valeurn"/>
        </xsd :restriction>
</xsd :simpleType>
```

  Chaque $valeur_i$ doit être unique.

- Gérer des expressions régulières. La facette *pattern* sera utilisée comme suit :

```
<xsd :simpleType  name="nom-de-mon-nouveau-type-simple">
    <xsd :restriction   base="xsd :le-type-simple-prédéfini">
        <xsd :pattern   value="l'expression régulière voulue"/>
    </xsd :restriction>
</xsd :simpleType>
```

  Par exemple : une expression régulière d'un e-mail sera : "(.)+@+(.)" c'est à dire : une

  chaîne de caractères suivie du caractère @ suivi d'une chaîne de caractères.

- Fixer la valeur minimale ou maximale d'un type numérique. Par exemple la facette *maxExclusive* sera utilisée comme suit :

```
<xsd :simpleType  name="nom-de-mon-nouveau-type-simple">
    <xsd :restriction   base="xsd :le-type-simple-prédéfini">
        <xsd :maxExclusive   value="valeur"/>
    </xsd :restriction>
</xsd :simpleType>
```

Cette syntaxe est utilisée pour fixer la valeur maximale d'un type. Pour fixer la valeur minimale d'un type nous utilisons à la place de *maxExclusive* la facette *minExclusive.*

La valeur des attributs des facettes doit être conforme aux types simples que nous voulons restreindre. L'exemple suivant provoque une erreur :

```
<xsd :simpleType   name="typeImpossible>
   <xsd:restriction   base="xsd:integer">
      <xsd :lentgh  value="3"/>
      <!- -Un entier de longueur 3, ça ne veut rien dire- ->
   </xsd :restriction>
</xsd :simpleType>
```

La liste détaillée de toutes les facettes ainsi que les types sur lesquels elles sont applicables est beaucoup plus consistantes que les exemples cités ci-dessus [27], [29].

2.9.1.2 Restriction de types complexes

Il est également possible d'appliquer la dérivation par restriction aux types complexes.
Un type complexe dérivé par restriction est semblable à son type de base, excepté le fait que ses déclarations dans son modèle de contenu sont plus limitées que celles du type de base. En réalité, les valeurs représentées par le nouveau type sont un sous-ensemble des valeurs du type de base, c'est également le cas pour les types simples [27], [30].
Pour restreindre un type complexe, nous pouvons faire appel à l'attribut *maxOccurs*.
D'où la syntaxe :

```
<xsd :complexType  name="nom-de-mon-nouveau-type-complexe">
   <xsd :complexContent>
      <xsd :restriction    base="le-type-complexe-de-base">
          <xsd :sequence>
             <xsd :element    name="nom1"   maxOccurs="val1"    type="type1"/>
          </xsd:sequence>
      </xsd:restriction>
   </xsd:complexContent>
<xsd:complextype>
```

2.9.2 Extension

2.9.2.1 Extension de types simples

Si nous avons un élément de type simple et que nous voulions lui rajouter un attribut, le type de cet élément ne deviendrait plus simple, il deviendrait complexe, la question qui se pose est : comment définit-on un type complexe basé sur un type simple ? La réponse à cette question est de dériver un type complexe à partir d'un type simple, ce mécanisme est appelé l'extension de types simples.
Cette méthode permet de créer des types complexes dont le contenu est un type *simple* et qui peuvent porter des attributs.
Sa syntaxe est :

```
<xsd :complexType  name="nom-du-nouveau-type-complexe">
  <xsd :simpleContent>
     <xsd :extension    base="xsd :le-type-simple-prédéfini">
        <xsd :attribute  name="nom-attr1"   type="type-attr1"/>
                         ¦
                         ¦
                         ¦
        <xsd :attribute  name="nom-attrn"   type="type-attrn"/>
      </xsd:extension>
  </xsd:simpleContent>
<xsd:complexContent>
```

L'élément *simpleContent* du langage XML Schéma indique que notre nouveau type contient uniquement un type simple et pas d'éléments fils. L'élément *extension* porte l'attribut *base* qui indique le type simple de base, les attributs *nom-attr*$_1$*…nom-attr*$_n$ étant des éléments fils de l'élément *extension* [27], [30].

2.9.2.2 Extension de types complexes

Alors que la dérivation par restriction de type complexe réduit la valeur du type à un sous-ensemble du type de base, la dérivation par extension rend le modèle de contenu du type complexe de base plus riche. Elle permet par exemple de rajouter des éléments ou des attributs ou même les deux à la fois, au modèle complexe de base.
Cette extension se fait comme suit:

```
<xsd :complexType   name="nom-du-nouveau-type-complexe">
  <xsd :complexContent>
     <xsd :extension  base="nom-du-type-complexe-de-base">
        <xsd :sequence>

          < !- -  les déclarations d'éléments rajoutés- - >

        </xsd :sequence>
         < !- -nous pouvons éventuellement rajouter des attributs- ->
             <xsd :attribute   name="nom-attr1"  type="type-attr1"  use=" la valeur
             adéquate1"    value="val1"/>
                    ¦
                    ¦
                    ¦
             xsd :attribute   name="nom-attrn"  type="type-attrn"  use="la valeur
             adéquaten"    value="valn"/>
     </xsd :extension>
  </xsd :complexContent>
</xsd :complexType>
```

Nous définissons des nouveaux types complexes à l'aide de l'élément *complexType,* en prenant soin de préciser que le modèle de contenu de ces types est complexe avec l'élément *complexContent*. Lorsqu'un type complexe est dérivé par extension, son modèle de contenu effectif est celui de base concaténé au modèle de contenu spécifié dans la dérivation. En effet, les deux modèles de contenu sont considérés par XML Schéma comme deux fils d'un groupe séquentiel [27], [30].

2.9.3 Utilisation des types dérivés dans les documents instances

XML Schéma nous autorise à déclarer un élément ayant un type de base et à utiliser des instances des types dérivés de ce type de base dans les documents instances. Toutefois, il est nécessaire d'identifier explicitement dans le document instance de quel type il s'agit (type de base ou type dérivé) grâce à l'attribut *type* préfixé par le préfixe associé à l'espace de noms des instances :*www.w3.org/2001/XMLSchema-instance*. Cet attribut doit être qualifié, de façon explicite ou implicite. Mais il est d'usage de le qualifier de façon explicite.
Notons également que l'on doit déclarer l'espace de nom des instances XML Schéma dans le document instance, et cela grâce à l'attribut *xmlns* , cet attribut est un attribut de l'élément racine *schéma.* Dans la plupart des cas le préfixe associé aux instances est *xsi* [27], [30].

2.9.4 Contrôler la dérivation de types

Dans la pratique, il peut être utile à un auteur de schéma de limiter la dérivation de certains types.
XML Schéma propose un mécanisme exhaustif de contrôle de la dérivation grâce à l'attribut *final* qui sera un attribut de l'élément *comlpexType* qui est le type complexe de l'élément auquel nous ne pouvons pas appliquer de dérivation. Ce mécanisme permet à un auteur de schéma de déclarer, pour un type simple ou complexe, que celui ci ne sera pas : soit dérivable par restriction *(final="restriction")*, soit dérivable par extension *(final="extension")*, soit encore pas du tout dérivable (*final="#all").*
Un autre attribut *finalDefault* porté par l'élément *schéma* a les mêmes effets que de préciser la valeur de l'attribut *final* à chaque définition de type.
Autre façon de contrôler la dérivation (d'un type simple) est l'attribut *fixed* (pouvant prendre la valeur *true* ou *false)*, lorsqu'il est porté par une facette, il empêche la dérivation du type simple lorsque sa valeur est *true* [27], [30].

2.9.5 Contrôler l'utilisation de types dérivés

Un utilisateur de schéma peut aussi contrôler l'utilisation de types dérivés avec l'attribut *block* qui selon sa valeur : *restriction, extension* ou *#all,* empêche l'apparition de types dérivés attendant le type de base. Cet attribut est porté par l'élément *complexType* qui définit le type de base.
Enfin, comme avec l'attribut *final*, il existe un attribut optionnel *blockDefault* porté par l'élément *schema* qui permet de spécifier une valeur par défaut du blocage pour tous les types du schéma [27], [30].

2.10 Conception avancée des schémas

2.10.1 Groupes de substitution

Les groupes de substitutions sont un mécanisme fourni par XML Schéma pour permettre à un élément d'être remplacé par d'autres éléments. De façon plus détaillée, des éléments peuvent faire parti d'un groupe spécial d'éléments qui sont dits *substituables* pour un élément particulier appelé élément *tête* (*head element)*.
Il faut souligner le fait que cet *élément tête* doit toujours être un élément déclaré de façon *globale*. Retenons également que les éléments membres d'un groupe de substitution doivent

être du même type que l'élément *tête*, ou éventuellement d'un type dérivé du type de l'élément *tête.*
Soit un élément déclaré dans XML Schéma comme suit :
<xsd :element name="nom-élt-tête" type="type-élt-tête"/>

Pour pouvoir utiliser un groupe de substitution, nous utilisons la syntaxe suivante :

<xsd :element name="nom-élt-subs$_1$" type="type-élt-tête" substitutionGroup="nom-élt-tête"/>
¦
¦
¦
<xsd :element name="nom-élt-subs$_n$" type="type-élt-tête" substitutionGroup="nom-élt-tête"/>

L'existence d'un groupe de substitution n'exige pas d'utiliser un quelconque des éléments de cette classe, ni ne présume l'utilisation de l'élément tête. Il fournit simplement un mécanisme pour permettre aux éléments d'être interchangeables [27], [30].

2.10.2 Eléments abstraits

L'abstraction d'un élément est en réalité un moyen d'en forcer la substitution par un autre élément.
Si un élément est déclaré abstrait dans un schéma, il ne peut apparaître dans le document instance. Un élément membre de son groupe de substitution doit être utilisé à sa place [27], [30].
La déclaration d'un élément abstrait doit être accompagné de la déclaration du groupe de substitution.
Sa syntaxe est :

<xsd :element name="nom-élt" type="type-élt" abstract="true"/>
<xsd :element name="nom-élt-subs$_1$" type="type-élt" substitutionGroup="nom-élt"/>
¦
¦
¦
<xsd :element name="nom-élt-subs$_n$" type="type-élt" substitutionGroup="nom-élt"/>

2.10.3 Types anonymes

XML Schéma nous autorise à utiliser des types de données *anonymes* déclarés *localement* dans le modèle de contenu de l'élément de type *anonyme* (en général, nous pouvons identifier les types anonymes en notant l'absence de l'attribut type simple ou complexe).
Cette méthode est plus optimale lorsque ces types ne sont pas référencés plusieurs fois ou ne possèdent pas beaucoup de contraintes. Les types *anonymes* peuvent être utilisés pour les types simples ou complexes [27], [30].

2.10.4 Réutilisation de schémas existants

2.10.4.1 Inclusion

Lorsqu'un schéma dépasse une taille critique, en général une page, il devient plus difficile à maintenir. Il est alors opportun de scinder ce schéma en plusieurs documents.

Pour inclure un schéma fichier1.xsd à l'espace de nom de fichier2.xsd nous utilisons l'instruction **<xsd :include schemaLocation="URI de fichier1.xsd"/>** dans le fichier2. L'espace de noms cible des deux schémas doit être le même, et l'attribut *schemaLocation* est obligatoirement porté par l'élément *include* [27].

2.10.4.2 Redéfinition

Le mécanisme d'inclusion nous permet d'utiliser des composants de schémas externes sans modification, alors que le mécanisme de redéfinition permet de redéfinir des types simples ou complexes issus de schémas externes.
Pour redéfinir un type, nous le dérivons par extension en donnant le même nom au nouveau type complexe, et cela doit absolument être à l'intérieur de l'élément *redefine* déclaré dans la balise suivante :
<xsd :redefine schemaLocation="chemin de l'élément à redéfinir">
Dériver un type en donnant le même nom au nouveau type sans l'élément *redefine* provoque une erreur.
Une fois qu'un type est redéfini, le nouveau type va servir de base à tous les composants du schéma qui l'utilise.
La redéfinition nécessite que les composants externes appartiennent au même espace de noms cible que le schéma qui les redéfinit. Les composants externes n'ayant pas d'espace de noms peuvent également être redéfinis, dans ce cas ils deviennent parti intégrante de l'espace de noms cible du schéma les redéfinissant [27].

2.10.4.3 Importation de types

Contrairement au mécanisme d'inclusion, le mécanisme d'importation permet d'utiliser des composants externes appartenant à de différents espaces de noms cibles, qui seront la valeur de l'attribut *namespace*, cet attribut sera suivi de l'attribut *schemaLocation* qui indique l'emplacement du schéma à importer.
Nous devons associer un préfixe à chaque espace de noms, par lequel sera référencé le type importé.
Seuls les éléments, attributs et types déclarés *globalement* peuvent être importés.
Les types peuvent aussi être importés comme types de base pour dériver de nouveaux types. Seuls les types complexes nommés peuvent être importés, les types locaux définis anonymement ne peuvent pas être importés.
Lorsque des composants de schémas sont importés de plusieurs espaces de noms, chaque espace de noms doit être identifié avec un élément *import* séparé. Les éléments *import* doivent apparaître comme les premiers fils de l'élément *schema*. De plus, chaque espace de noms doit être associer à un préfixe en utilisant une déclaration standard d'espace de noms. Ce préfixe sera utilisé pour qualifier les références à n'importe quel composant appartenant à cet espace de noms [27].

2.10.5 Annotation

XML Schéma fournit trois éléments pour documenter un schéma. Le premier élément, *annotation*, qui peut être fils de l'élément *shema* ,s*impleType, complexType* ou *attribute,* peut contenir soit des informations et des commentaires à l'intention d'un lecteur humain signalées par le sous-élément *documentation*, soit des informations destinées à une application, telle une feuille de style, contenue dans le sous-élément *appInfo* [27], [30].

Nous pouvons par exemple fournir des informations de copyright aux lecteurs de notre schéma de la façon suivante :

<xsd :annotation>
<xsd :documentation>Copyright 2001 Grégory CHALAZON Joséphine LEMOINE, Tous droits réservés </xsd :documentation>
</xsd :annotation>

2.10.6 Valeur *nil*

En général, l'absence d'un élément n'a pas de signification particulière : cela peut signifier au choix que l'information est inconnue, ou qu'elle est non applicable, ou que l'élément est absent pour une quelconque autre raison. Parfois, il est souhaitable de pouvoir représenter cet élément plutôt que de l'omettre.
Dans XML Schéma, nous déclarons cet élément à l'aide de l'attribut *nillable* ayant la valeur *true* .
Dans le document instance, cet élément portera l'attribut *xsi :nil* à *true* et ne peut contenir aucun élément fils, en revanche il peut porter des attributs.
L'attribut *nil* fait parti de l'espace de noms des instances XML Schéma, et c'est pour cela qu'il est explicitement qualifié à l'aide du préfixe *xsi*.
Le mécanisme *nil* ne s'applique qu'aux contenus des éléments et pas à la valeur de leurs attributs [27], [30].

2.11 Contraintes d'unicité et contraintes référentielles

Les DTD fournissent déjà un mécanisme assurant l'unicité, au moyen de l'attribut ID et de ses attributs associés IDREF et IDREFS. Bien que ce mécanisme ait été repris pour des raisons de compatibilité, XML Schéma introduit de nouveaux mécanismes plus souples et plus puissants.

Tout d'abord, ces mécanismes peuvent s'appliquer aussi bien à des éléments qu'à des attributs de n'importe quel type. De plus, grâce à XPATH, XML Schéma permet de spécifier la portée de la contrainte d'unicité. Enfin, il est possible de créer des clés et des références à partir de combinaisons d'éléments ou d'attributs, ce qui est impossible avec les DTD.

2.11.1 Contraintes d'unicité

Supposons que nous voulons intégrer des données à notre document XML (nous verrons dans le prochain chapitre qu'un document XML est une base de données), comme toute base de données, notre document doit également avoir un champ représentant sa clé qui permet d'identifier de façon unique un enregistrement.
Si la clé n'est pas mise en évidence dans un document XML, ce dernier peut devenir corrompu à cause d'enregistrements redondants après une intégration de données.
La clé doit donc être déclarée dans un schéma grâce à l'élément *unique* qui porte comme fils deux éléments : *selector* (qui comporte un attribut *xpath* dont la valeur indique le chemin des éléments –constituant l'enregistrement- représentés par la clé) et *field* (qui indique au moyen de son attribut *xpath* le nom de l'attribut représentant la clé). Ainsi, nous pouvons être assuré que tout document instance validé par un processeur XML pourra être directement intégré –à la volée- à notre base de données.

Ce mécanisme est réalisable grâce aux balises suivantes :

```
< !- - ajout de l'attribut représentant la clé- ->
<xsd :attribute      name='nom-clé'     type='type-clé'     use='required' />
….
<!- - contraintes d'unicité- ->
<xsd:unique    name='clef-enregistrement'>
  <xsd:selector    xpath='chemin des éléments de l'enregistrement' />
  <xsd :field        xpath='@nom-clé' />
</xsd:unique>
```

2.11.2 Contraintes référentielles

Dans certains documents XML, nous pouvons avoir besoin de déclarer une clé étrangère qui va relier le document à un autre et elle doit être munie d'une contrainte d'intégrité référentielle.

Découvrons à présent la syntaxe permettant de spécifier une contrainte d'intégrité référentielle :

```
< !- - Déclaration de l'élément représentant la clé étrangère- ->
<xsd :element    name='nom-clé étrangère'   type='type-clé étrangère' />

< !- - contraintes d'intégrité référentielle- ->
<xsd :key    name='clef-enregistrement de l'autre document'>
  <xsd:selector  xpath='chemin des éléments de l'enregistrement de l'autre document'/>
  <xsd :field        xpath='@nom-clé de l'autre document' />
</xsd :key>


<xsd :keyref  name='nom-clé  étrangère'  refer='clef-enregistrement  de  l'autre
docuement' >
  <xsd:selector  xpath='chemin des éléments de l'enregistrement de notre document'/>
  <xsd :field        xpath='./nom-clé étrangère' />
</xsd :keyref>
```

En résumé, nous avons déclaré l'attribut *nom-clé de l'autre document* en tant que clé de l'autre document (en utilisant l'élément *key*), ensuite nous avons utilisé l'élément *keyref* pour indiquer une référence à cette clé.

2.12 Présentation des documents XML

Contrairement au HTML, XML ne propose aucun élément ou attribut de mise en page. Les descriptions sémantiques et physiques ont été séparées pour résoudre des problèmes tels que la dépendance à des browsers Web. La réalisation physique d'un document se fait grâce à la création d'une feuille de style où sont définies les règles de mise en page pour chaque type d'élément.
Le langage CSS (*Cascading Style Sheets)* a été conçu pour la spécification d'une feuille de style. Sa première version a été publiée en 1996 (CSS-1) et la deuxième version en 1998 (CSS-2).
La structure physique d'un document sera calquée sur sa structure logique en utilisant une feuille de style CSS. Cette dernière se compose de règles de style qui peuvent s'appliquer à un ou plusieurs éléments ou même à une partie d'un élément, ceci sera spécifié par le sélecteur. La présentation physique sera décrite par une ou plusieurs propriétés, (CSS définit 122 propriétés). La forme d'une règle est : **sélecteur{ propriété$_1$:valeur$_1$; propriété$_2$:valeur$_2$; ...}**. Si une règle est appliquée à un élément, elle sera appliquée à tous ses fils par défaut sauf si le contraire est spécifié. Les règles peuvent être importés d'une autre feuille de style. Enfin, (CSS-2) a introduit la notion de type de média car il peut spécifier pour un document sa présentation en vue de son affichage sur écran, de son impression ...etc.

Un autre langage, XSL (*eXtensible Stylsheet Langage)*, offre de nouvelles possibilités par rapport au CSS. Il permet de modifier la structure du document, c'est à dire la transformation de l'arbre source. Une des applications les plus courantes est de générer une version HTML d'un document XML. Une feuille de style XSL peut, en outre, définir un arbre contenant un élément représentant une table des matières ou une indexation ce qui apportera une clarté au document. Un document XSL, c'est à dire, une feuille de style XSL, contient des modèles. Un processeur XSL compare les éléments d'un document XML d'entrée aux modèles de la feuille de style.

Quand il trouve un modèle adapté, il copie le contenu du modèle dans l'arbre de sortie. Une fois cette étape accomplie, il peut sérialiser l'arbre de sortie en un document XML ou un autre format, comme du texte brut ou du HTML.
Certains attributs permettent d'utiliser un élément plusieurs fois dans le même document mais sous plusieurs formes (formaté différemment) [24], [25].

2.13 Les API (Application Programming Interface)

Pour pouvoir manipuler un document XML en lecture et/ou en écriture il est nécessaire de disposer d'outils spécifiques.
XML permet donc de définir un format d'échange selon les besoins de l'utilisateur et offre des mécanismes pour vérifier la validité du document produit. Il est donc essentiel pour le receveur d'un document XML de pouvoir extraire des données du document. Cette opération est possible à l'aide d'un outil appelé : analyseur (en anglais : parser souvent francisé en : parseur) [28].
Toute application qui doit traiter des entités XML est interfacée avec un parseur à travers une API. Une API consiste en un ensemble de composants logiciels (interfaces, classes, méthodes, variables,…) pouvant être utilisé par un programme, dans le but de réaliser ou de faciliter un certain traitement. Il existe différentes interfaces de programmations (API) pour différents langages de programmation, permettant de travailler sur un document XML. Les deux

principales catégories d'API disponibles sont **SAX** et **DOM** qui correspondent à deux modèles de traitements différents de documents.

2.13.1 DOM

DOM (Document Object Model, en français : modèle objet de document) est une spécification du W3C définissant la structure d'un document sous forme d'une hiérarchie d'objets (c'est-à-dire sous forme arborescente) afin de simplifier l'accès aux éléments constitutifs du documents.
Plus exactement DOM est un langage normalisé d'interface indépendant de toute plate-forme et de tout langage, permettant à une application après avoir parsé le document XML de parcourir la structure du document et d'agir dynamiquement sur celui-ci. Ainsi, JavaScript et ECMAScript utilisent DOM pour naviguer au sein du document HTML, ce qui leur permet par exemple de pouvoir récupérer le contenu d'un formulaire, le modifier, consulter ses éléments…etc [28]

2.13.2 SAX

Dans l'API SAX (Simple API for XML) le parseur lit le flot de données XML en en entrée et reconnaît et interprète les marques de balisage au fur et à mesure qu'ils les rencontrent. Chaque construction reconnue est immédiatement signalée et passée à l'application. Ce modèle est appelé : traitement dirigé par les événements, car chaque événement de l'analyse du flot de données (par exemple, reconnaissance d'un début de d'élément ou de la fin de l'élément) est reporté à l'application par une fonction de renvoi (callback). L'application implémente une fonction de prise en charge (un handler) pour chaque type d'événement.

Soit le document XML suivant :
<personne>
<nom>Levert</nom>
<prenom>Albert</prenom>
</personne>

Une interface événementielle telle que l'API SAX permet de créer des événements à partir de la lecture du document ci-dessus. Les événements générés sont :
Start document
Start element :personne
Start element :nom
Characters : Levert
End element : nom
Start element :prenom
Characters : Albert
End element : prenom
End element : personne
End document

Ainsi une application basée sur SAX peut générer uniquement les éléments dont elle a besoin sans avoir à construire en mémoire une structure contenant l'intégralité du document [28].

2.13.3 Comparaison entre SAX et DOM

Les deux modèlent offrent avantages et inconvénients. En pratique, ils se prêtent plus ou moins bien à certains traitements. Le modèle de compilation d'arbre (DOM) est évidemment coûteux si le document est volumineux. On conçoit bien que si le traitement recherché est de trouver tous les éléments qui contiennent une certaine chaîne de caractères, le modèle événementiel (SAX) sera beaucoup plus économique et efficace. L'application se contentera de vérifier la présence de la chaîne de caractères cible dans le contenu de type texte de chaque élément, puis de représenter dans ses structures de données propres le résultat obtenu, par exemple le contenu complet de l'élément. L'arbre total représentant le document entier n'est ainsi jamais construit en entier à un instant donné, ni à l'échelon du parseur ni à celui de l'application. Si le document représente 2 ou 3 Mo de caractères, on conçoit l'économie de traitement réalisé et le gain en temps qui en résulte.

En revanche, si l'application a besoin d'accéder aléatoirement à des composants de l'arbre, il est clair que le modèle fondé sur la représentation de l'arbre complet du document (DOM) sera préférable. Les applications les plus typiques qui exploitent le DOM sont les éditeurs interactifs et les navigateurs XML [28].

2.14 Les liens dans XML

Nous avons montré dans les DTD comment définir des liens hypertextuels internes à un document à l'aide des attributs de type *ID* et *IDREF*. XML a été conçu comme un langage permettant la distribution de documents et de données sur le Web, et il est clair que ce mécanisme de liens internes est tout à fait insuffisant pour l'immense majorité des documents présents sur la toile. Il existe plusieurs types de liens XML qui permettent de construire un modèle de toile très riche.

Les liens XML (XLink en abrégé) peuvent se définir extérieurement aux entités qu'ils relient : un lien reliant un élément X d'un document D_1 à un fragment F d'un document D_2 pourra être un élément contenu dans un document D_3 . Il est donc possible de définir des liaisons entre plusieurs documents.

Grâce à XLink, tous les éléments peuvent être des liens (en HTML, seul l'ancre <A> pourrait représenter un lien). Ceci est dû en grande partie au XML, qui permet à l'utilisateur d'établir son propre jeu de balises.

Un autre mécanisme particulier, le pointeur XML ou XPointer, crée des liens vers des positions arbitraires dans un seul document, ces positions peuvent être absolues (Exemple : l'élément ayant l'attribut ID= ′233′) ou relatives (Exemple : le paragraphe suivant la seconde occurrence de la chaîne xyz dans une section de niveau 3), ainsi nous pouvons indexer des positions arbitraires dans un document XML [28]. Il est également possible grâce à XPointer de créer des liens ayant plusieurs cibles.

Chapitre3: XML et les Bases de Données

3.1 Introduction

Après avoir vu en détail dans les chapitres précédents les avantages du langage XML et l'apport des schémas XSD pour la construction de documents XML sûrs et fiables, et aussi l'utilité d'avoir un format pivot unique pour l'échange d'information entre plusieurs sources de données, nous allons dans le présent chapitre nous intéressé aux méthodes proposées par certains logiciels pour permettre la transformation d'une base de données (base de production ou base externe) quelconque (relationnelle ou orientée objet) vers une base de données XML, car en effet un document XML peut être considéré comme étant une base de données qui exploite pleinement les avantages du XML [16].

Nous pouvons utiliser XML partout où l'on trouve de la donnée, c'est à dire partout dans les systèmes d'informations et plus particulièrement dans les bases de données.

Le rôle des bases de données est fondamental dans les systèmes d'information. Les principales catégories sont les bases de données objets et relationnelles. L'avènement des technologies XML a été suivi de l'arrivée des bases de données natives XML, et d'un nouveau débat : Faut-il un nouveau type de base de données pour gérer le XML?

Le langage XML est en passe de devenir le standard utilisé systématiquement par les entreprises –et notamment dans les Data Warehouses- pour échanger des données dans des documents structurés, que ce soit en interne, avec des partenaires commerciaux ou dans des applications accessibles à tous sur internet. Toutefois, les données échangées sont souvent stockées dans une base de données relationnelle, orientée objet ou qui ont un format différent du XML. Il faut donc mettre en place un processus de traduction des données du format actuel de la base en XML (et éventuellement du XML au format utilisé pour le traitement des données).

Les clients des entreprises adoptent le stockage XML pour sa capacité à respecter et conserver le document dans son intégralité. Si, par exemple, les informations d´une facture sont éclatées dans une base de données relationnelle, l´intégrité du document original est perdue. Il n´est donc plus exploitable en cas de litige.

3.2 Un document XML est-il une base de données ?

Un document XML peut être considéré comme une base de donnés uniquement au sens le plus strict de l'expression. A savoir, une collection de données. XML en cela n'est guère différent des autres fichiers, car après tout, tous les fichiers contiennent d'une certaine manière des données. En tant que format de "base de données", XML présente cependant certains avantages. Il est portable puisque codé en Unicode, et il peut décrire les données sous la forme d'un arbre ou d'un graphe.

Une question plus intéressante consiste à se demander si XML et les technologies qui lui sont associées constituent une "base de données" dans un sens plus large, c'est-à-dire au sens d'un système de gestion de base de données (SGBD). La réponse à cette question est : « XML est une sorte de SGBD ». Dans le sens d'une réponse positive, XML fournit plusieurs des caractéristiques que l'on retrouve dans les bases de données : le stockage (les documents XML), les schémas (DTD, Schémas XML), des langages de requête (XQuery, XPath, XQL, XML-QL, QUILT, etc.), des interfaces de programmation (SAX, DOM, JDOM), et ainsi de

suite, sans oublier que ces tous ce sont ces caractéristiques qui ont fait la force des bases de données relationnelles. Dans le sens d'une réponse négative, de nombreuses caractéristiques présentes dans les bases de données lui font défaut : un stockage efficace, les index, la sécurité, les transactions, l'accès multi-utilisateurs, les déclencheurs (*triggers*), les requêtes sur plusieurs documents … etc.

Il existe d'autres exemples plus sophistiqués de collections de données pour lesquelles un document XML pourrait être utilisé en tant que base de données, nous citerons à titre d'exemple les listes de contacts personnels (noms, numéros de téléphone, adresses, etc.) ou tout document qui pourrai servir de preuve légale [32].

3.3 Contenus XML orientés Données et Document

Le facteur le plus important en ce qui concerne le choix d'une base de données est peut-être de savoir si nous utiliserons la base pour stocker des données ou des documents. Est-ce que, par exemple, XML est utilisé simplement en tant que vecteur de données entre la base et une application? Ou bien XML est-il exploité intégralement, à la manière des documents au format XHTML ou DocBook ? Il s'agit là habituellement d'une question pratique, mais elle est d'importance, car tous les contenus orientés données partagent un certain nombre de caractéristiques (il en est de même de leur côté des contenus orientés document) et ces caractéristiques influencent la manière dont XML est stocké dans la base de données. Les deux sections suivantes examinent ces caractéristiques [32].

3.3.1 Les contenus orientés Données

Les contenus *orientés données* sont caractérisés par une structure assez régulière, des données qui présentent une granularité fine (c'est-à-dire que la plus petite unité indépendante de donnée est située au niveau d'un élément de type PCDATA unique ou d'un attribut), et peuvent ou pas du tout avoir des contenus mixtes. L'ordre dans lequel les éléments fils d'un même parent et les PCDATA apparaissent n'est en général pas significatif, excepté lorsque l'on valide le document au sens XML

A titre d'exemple, l'ordre de ventes suivant est *orienté données* :

```
<OrdreDeVentes NumeroOrdreDeVentes="12345">
    <Client NumeroClient="543">
      <NomClient>ABC Industries</NomClient>
      <Rue>123 Main St.</Rue>
      <Ville>Chicago</Ville>
      <Etat>IL</Etat>
      <CodePostal>60609</CodePostal>
    </Client>
    <DateOrdre>981215</DateOrdre>
    <Item NumeroItem="1">
      <Lot NumeroLot="123">
        <Description>
          <p><b>Turkey wrench:</b><br />
          Stainless steel, one-piece construction,
          lifetime guarantee.</p>
        </Description>
```

```
        <Prix>9.95</Prix>
      </Lot>
      <Quantite>10</Quantite>
    </Item>
    <Item NumeroItem="2">
      <Lot NumeroLot="456">
        <Description>
          <p><b>Stuffing separator:<b><br />
          Aluminum, one-year guarantee.</p>
        </Description>
        <Prix>13.27</Prix>
      </Lot>
      <Quantite>5</Quantite>
    </Item>
</OrdreDeVentes>
```

En plus de contenus aussi manifestement *orientés données* que l'ordre de ventes ci-dessus, de nombreux documents riches en texte sont également *orientés données*. Considérons par exemple une page du site Amazon.com affichant des informations à propos d'un livre. Bien que la page soit largement constituée de texte, la structure de ce texte est tout à fait régulière : la majeure partie est commune à toutes les pages qui décrivent des livres, et chaque portion de texte spécifique possède une taille limitée. Ainsi, la page peut être construite à partir d'un contenu *orienté données* simple contenant l'information relative à un livre unique ; ce contenu est retrouvé dans la base de données et une feuille de style XSL y ajoute les zones fixes. En règle générale, tout site Web actuel construisant dynamiquement du code HTML en remplissant un modèle à l'aide de données extraites d'une base peut probablement être remplacé par une série de documents XML *orientés données* et une ou plusieurs feuilles de style XSL [32].

Considérons par exemple le document suivant qui décrit un vol :

```
<InformationsVol>
    <Compagnie>ABC Airways</Compagnie> propose <Nombre>trois</Nombre>
    vols quotidiens sans escales depuis <Depart>Dallas</Depart> à destination de
    <Destination>Fort Worth</Destination>. Les heures de départ sont
    <HeureDepart>09:15</HeureDepart>, <HeureDepart>11:15</HeureDepart>,
    et <HeureDepart>13:15</HeureDepart>. Les arrivées interviennent une minute
plus                                                                    tard.
</InformationsVol>
```

Il pourrait être généré à partir du document XML suivant et d'une simple feuille de style :

```
    <Vols>
        <Compagnie>ABC Airways</Compagnie>
        <Depart>Dallas</Depart>
        <Destination>Fort Worth</Destination>
        <Vol>
          <HeureDepart>09:15</HeureDepart>
          <HeureArrivee>09:16</HeureArrivee>
      </Vol>
```

```
    <Vol>
      <HeureDepart>11:15</HeureDepart>
      <HeureArrivee>11:16</HeureArrivee>
    </Vol>
    <Vol>
      <HeureDepart>13:15</HeureDepart>
      <HeureArrivee>13:16</HeureArrivee>
    </Vol>
  </Vols>
```

3.3.2 Les contenus orientés Document

Les contenus *orientés document* sont habituellement conçus pour être utilisés par des humains. Les livres, messages électroniques, annonces, ainsi que presque toutes les pages XHTML écrites à la main constituent des exemples de tels documents. Ils sont caractérisés par une structure moins régulière ou même franchement irrégulière, des données qui présentent une granularité plus grande (c'est-à-dire que la plus petite unité indépendante de donnée peut être située au niveau d'un élément mêlant différents contenus, voire même au niveau du document lui-même), et beaucoup de contenus mixtes. L'ordre dans lequel les éléments fils d'un même parent et les PCDATA apparaissent est presque toujours significatif.

Le document suivant, par exemple, décrit un produit et il est *orienté document* :

```
<Produit>
  <Intro>
  The <ProductName>Turkey Wrench</ProductName> from <Developer>Full
  Fabrication Labs, Inc.</Developer> is <Summary>like a monkey wrench,
  but not as big.</Summary>
  </Intro>
  <Description>
  <Para>The turkey wrench, which comes in <i>both right- and left-
  handed versions (skyhook optional)</i>, is made of the <b>finest
  stainless steel</b>. The Readi-grip rubberized handle quickly adapts
  to your hands, even in the greasiest situations. Adjustment is
  possible through a variety of custom dials.</Para>
  <Para>You can:</Para>
  <Liste>
  <Item><Link URL="Order.html">Order your own turkey wrench</Link></Item>
  <Item><Link URL="Wrenches.htm">Read more about wrenches</Link></Item>
  <Item><Link URL="Catalog.zip">Download the catalog</Link></Item>
  </Liste>
  <Para>The turkey wrench costs <b>just $19.99</b> and, if you
  order now, comes with a <b>hand-crafted shrimp hammer</b> as a
  bonus gift.</Para>
 </Description>
</Produit>
```

3.3.3 Données, Documents et bases de données

La distinction entre contenus *orientés données* et contenus *orientés document* n'est pas toujours claire en pratique. Un contenu *orienté données* comme une facture par exemple peut contenir aussi des données de granularité forte et irrégulièrement structurées telles que des descriptions. Et inversement, un contenu *orienté document* comme un manuel utilisateur peut contenir des données de granularité fine et régulièrement structurées, telles que le nom de l'auteur ou une date de révision. Les documents juridiques ou médicaux constituent aussi d'autres exemples - ils sont écrits sous forme de prose mais contiennent des parties distinctes telles que des dates, des noms, des procédures, et doivent souvent être stockés dans leur intégralité pour des raisons légales.

En dépit de cette imprécision, la caractérisation de contenus comme *orientés données* ou *orientés document* nous aidera à décider du genre de base de données à utiliser. En règle générale, les documents *orientés données* sont stockées dans une base traditionnelle, qu'elle soit relationnelle ou orientée objet. Dans ce dernier cas, la base de données est qualifiée de *compatible XML* (*XML-enabled*). Les documents *orientés document* quand à eux sont stockés dans une *base XML native*, c'est-à-dire une base conçue spécialement pour stocker du XML [32].

3.4 Transformation vers une Base de Données XML (orientée Données)

Dans le but d'échanger des données entre les documents XML et une autre base de données, il est nécessaire de faire correspondre le schéma du document XML (c'est-à-dire la DTD ou les Schémas XML) avec le schéma de la base de données. Le logiciel de transfert de données est alors construit au dessus de cette correspondance [13], [14].
Dans ce cas, la structure du document doit coïncider exactement avec la structure attendue par la correspondance réalisée [32].

Les correspondances entre les schémas de documents et les schémas de bases de données sont effectuées sur les types des éléments, les attributs et le texte. Elles omettent la plupart du temps la structure physique (comme les entités, les sections CDATA et les informations concernant l'encodage) et certaines structures logiques (telles que les instructions de traitement, les commentaires, ainsi que l'ordre dans lequel les éléments et les PCDATA apparaissent dans une filiation). Ceci est d'ailleurs raisonnable car la base et l'application sont uniquement concernées par les données se trouvant à l'intérieur du document XML. Dans l'exemple évoqué précédemment concernant un ordre de ventes, il n'est pas important que le numéro du client soit stocké dans une section CDATA, une entité externe, ou directement dans une PCDATA, et il importe peu également que le numéro du client soit stocké avant la date de l'ordre ou après celle-ci [32].

3.4.1 Transformation d'une Base de Données Relationnelle vers une Base de Données XML

Cette transformation est aussi appelée **la correspondance basée sur des tables,** elle est utilisée par de nombreux logiciels intermédiaires (*middleware*) [13], [14] qui transfèrent les données entre un document XML et une base de données relationnelle. Elle modélise les documents sous la forme d'une table unique ou comme un ensemble de tables. Cela signifie que la structure d'un document XML va être comme suit :

```
<database>
    <table>
      <row>
        <column1>...</column1>
        <column2>...</column2>
        ...
      </row>
      <row>
        ...
      </row>
      ...
    </table>
    <table>
      ...
    </table>
    ...
</database>
```

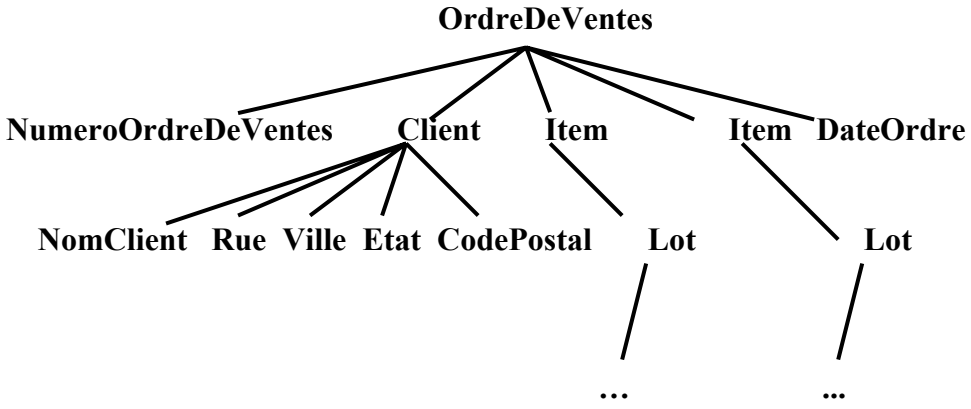
Le schéma ci-dessus montre bien la facilité de transformation d'une base de données relationnelle (caractérisée par ses lignes et colonnes) vers une base de données XML qui va avoir des balises correspondant aux balises <row> (qui correspond à la ligne) et <column> (qui correspond à la colonne).
L'élément <database> et les éléments supplémentaires <table> n'existent pas dans le cas où la modélisation est effectuée à l'aide d'une table unique.
Selon le logiciel, il peut être possible de spécifier si la colonne de données est stockée en tant qu'éléments fils ou comme attributs, ainsi que les noms à utiliser pour chaque élément ou attribut. De plus, les produits utilisant la correspondance basée sur des tables incluent souvent de manière optionnelle des méta-données de table ou de colonne soit au début du document, soit comme attribut de chaque élément table ou colonne. Notons que le terme "table" est habituellement interprété de manière vague. Autrement dit, quand on transfère des données depuis une base vers un document, une "table" peut être constituée par n'importe quel ensemble de résultats, et quand on transfère des données depuis un document XML vers la base, une "table" peut être une véritable table ou une vue.
La correspondance basée sur des tables est utile pour sérialiser des données relationnelles, comme par exemple pour transférer des données entre deux bases de données relationnelles. Son inconvénient évident est qu'elle ne peut pas être utilisée pour un document qui n'est pas – ou ne peut pas être- conforme au schéma exposé ci-dessus [32], [52].

3.4.2 Transformation d'une Base de Données Orientée Objet vers une Base de Données XML

Cette transformation est aussi appelée **la correspondance basée sur un modèle objet**, elle est utilisée par toutes les bases de données orientées objet compatibles XML (*XML-enabled*) et quelques produits intermédiaires (*middleware)*. Les données du document sont alors modélisées comme un arbre d'objets spécifiques aux données. Ce modèle est très facile à réaliser du moment qu'un document XML a une structure arborescente. Dans ce modèle, les classes d'une base de données orientée objet sont généralement modélisés par les types d'éléments possédant des attributs, les contenus d'éléments ainsi que les contenus mixtes (les *types d'éléments complexes)* provenant du document XML. Les propriétés scalaires

correspondent aux types d'éléments contenant seulement des PCDATA (les *types d'éléments simples*), les attributs et les PCDATA.

A titre d'exemple, le document ordre de ventes décrit précédemment pourrait être modélisé comme un arbre d'objets composé de quatre classes - OrdreDeVentes, Client, Item, Lot – et plusieurs propriétés scalaires – NumeroOrdreDeVentes, DateOrdre, CodePostal, NomClient, Rue, Ville, Etat- tel que le montre le schéma suivant :



Les données sont transférées entre les objets et le document XML.

La manière dont la correspondance basée sur un modèle objet est supportée varie d'un produit à l'autre. Par exemple :

- Tous les produits supportent la correspondance des types d'éléments complexes avec les classes ainsi que des types d'éléments simples et des attributs avec les propriétés scalaires.
- La plupart des produits permettent de spécifier l'ordre dans lequel les éléments fils apparaissent sous leur parent, bien qu'il soit impossible de cette manière de construire plusieurs modèles de contenu. (L'ordre des éléments fils est toutefois important quand nous souhaitons valider le document).
- Quelques produits permettent de faire correspondre des types d'éléments complexes à des propriétés scalaires. Ceci est utile lorsque le type d'élément contient des contenus mixtes et que l'utilisateur ne souhaite pas les décomposer, dans ce cas il est plus intéressant de voir le type complexe comme une seule propriété que de la décomposer en ses composantes.

3.5 Transformation vers une Base de Données XML (orientée Document)

Il se peut que la base de données contienne du texte dans une variable de type BLOB comme par exemple un document légal et que nous sommes obligés à ce moment là de garder ce document tel qu'il est. Après le transfert, ces données feront partie d'un document XML qui n'est rien d'autre qu'une base de données appelé base de données XML native.

3.5.1 Définition d'une Base de Données XML Natives

Une base de données XML native est une base spécialement conçue pour stocker du XML ou toute base pouvant avoir un contenu document.

Comme toutes les autres bases, elles possèdent des fonctionnalités telles que les transactions, la sécurité, les accès multi-utilisateurs, un ensemble d'APIs, des langages de requête… etc. Bien que ces derniers points restent encore à compléter et à perfectionner car ils sont encore à leur début. La seule différence par rapport aux autres bases, c'est qu'elles sont basées sur XML.
Les bases de données XML natives sont plus utiles pour le stockage des contenus *orientés document*, en raison du fait qu'elles préservent certains aspects tels que l'ordre interne du document, les instructions de traitement, les commentaires, les sections CDATA, l'utilisation des entités… etc, ce que ne font pas les bases qui sont seulement compatibles XML. En outre, les bases XML natives supportent les langages de requête XML qui permettent de poser des questions telles que : « Donnez-moi tous les documents dans lesquels le troisième paragraphe après le début d'une section contient un mot en caractères gras. » De telles recherches sont manifestement difficiles à formuler dans un langage tel que SQL.

Le document XML est l'unité fondamentale du stockage dans une base de données XML native, tout comme une ligne d'une table constitue l'unité fondamentale du stockage dans une base relationnelle.

Les bases de données XML natives disponibles aujourd'hui en sont encore à leurs débuts. La plupart des utilisateurs qui ont déjà adopté ces technologies travaillent dans les secteurs de la finance ou de la production, et utilisent principalement les bases de données XML natives pour accélérer les vitesses de transaction. Deux des bases de données XML natives les plus mûres sont Software AG Tamino et Ipedo XML Database [32].

3.5.2 Les langages de requêtes

Toutes les bases XML natives supportent un ou plusieurs langages de requêtes. Les plus populaires d'entre eux sont XPath (avec des extensions permettant des recherches sur des documents multiples), XQL et XQuery [32].

3.5.3 Mises à jour et effacements

Les bases XML natives possèdent une grande variété de stratégies pour réaliser les mises à jour et les effacements de documents. Cela va d'un simple remplacement ou effacement du document existant jusqu'aux modifications effectuées à travers un arbre DOM actif ou aux langages qui spécifient comment modifier des fragments de document. En règle générale, chaque produit permettant de modifier des fragments de document possède son propre langage, bien qu'un certain nombre de produits supportent le langage XUpdate du groupe XML :DB Initiative. Néanmoins, les possibilités de mise à jour resteront vraisemblablement incomplètes dans un avenir proche, car il s'agit encore d'un domaine d'étude et de recherche pour l'industrie et l'université [32].

3.5.4 Transactions, verrouillages et accès concurrentiels

Pratiquement toutes les bases XML natives supportent les transactions (et aussi les annulations de transactions : *rollbacks*). Le verrouillage est cependant réalisé le plus souvent au niveau du document dans son intégralité plutôt qu'au niveau des fragments du document, et ainsi, les accès multi-utilisateurs concurrents peuvent être relativement lents. La question de savoir si cela pose un problème dépend de l'application et de ce qui constitue un "document".

Si, par exemple, un guide d'utilisateur a été scindé en plusieurs chapitres individuels, chacun de ceux-ci constituant un document, alors les problèmes d'accès concurrents sont probablement réduits car il est peu probable que deux rédacteurs mettent à jour le même chapitre au même moment. Si, par contre, toutes les données clients d'une entreprise sont stockées dans un même document (ce qui serait très mal conçu), alors le verrouillage au niveau du document se révèlera très certainement désastreux.

3.5.5 Les index

Toutes les bases XML natives supportent l'indexation des valeurs des éléments et des attributs. Les index sont utilisés pour accélérer les recherches, comme c'est le cas pour les bases de données non XML**.**

3.5.6 L'intégrité référentielle

L'intégrité référentielle se réfère à la validité des pointeurs vers des données liées et il s'agit là d'un aspect indispensable de la tâche consistant à maintenir l'état consistant d'une base. Il n'est pas bon en effet qu'un ordre de ventes contienne un numéro de client qui ne corresponde à aucun enregistrement client; le service des livraisons ne sait pas où envoyer les articles achetés et le service de facturation ne sait pas où envoyer la facture.
Dans une base relationnelle, l'intégrité référentielle signifie que l'on s'assure que les clés étrangères pointent bien sur des clés primaires valides - autrement dit- que l'on vérifie que la ligne de clé primaire correspondant à une clé étrangère existe bien. Dans une base XML native, l'intégrité référentielle signifie que l'on s'assure que les "pointeurs" dans un document XML pointent bien sur des documents ou des fragments de documents valides.
Les pointeurs apparaissent sous plusieurs formes dans un document XML: des attributs ID/IDREF, des champs key/keyref (tels qu'ils sont définis dans les Schémas XML), des XLinks, et différents mécanismes propriétaires. Ces derniers incluent des éléments et des attributs de "référencement" propres au langage.

3.6 La génération de schémas XML à partir de schémas relationnels/orienté objet et vice-versa

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

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

5. Pour chaque élément fils complexe, lier la table du type d'élément parent à la table du type de l'élément fils à l'aide de la clé primaire de la table parente.

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

1. Créer un type d'élément par table.
2. Pour chaque colonne de cette table qui ne soit pas une clé et pour la (les) colonne(s) correspondant à la clé primaire, ajouter un attribut au type d'élément ou ajouter un élément fils de type PCDATA.
3. Pour chaque clé étrangère, ajouter un élément fils au contenu du modèle et traiter récursivement la table de la clé étrangère.

3.7 Schémas des Bases de Données XML Natives

XML –à lui tout seul- ne supporte pas la notion de *type de données*. Toute donnée d'un document XML est du texte, même si elle représente un autre type comme une date ou un nombre entier. Mais, tous les types des éléments et attributs sont bien décrits dans le schéma associé au document XML. Malheureusement et contrairement à une base de données relationnelle ou orientée objet, un document XML n'est pas toujours accompagné du schéma qui le décrit, néanmoins, il existe des logiciels permettant de générer le schéma correspondant à un document XML, ces logiciels ce basent principalement sur l'association de chaque élément ou attribut à son type et par la suite ils peuvent construire aisément son schéma [13], [14].

De plus, pour les documents HTML par exemple qui sont très répandus sur le Web, il très facile d'en créer une version XML (grâce aux API telles que SAX ou DOM), et un schéma XSD associée, car il suffit de donner à chaque élément le type qui lui est associé, et le seul type utilisé dans les documents HTML est le type chaîne de caractères.

3.8 Comparaison entre une Base de Données Relationnelle et une Base de Données XML native

Après avoir passer en revue les différents aspects des bases de données XML natives, nous allons établir un tableau comparatif entre ce qu'offrent les bases de données relationnelles et les bases de données XML natives :

Critères de comparaison	BD Relationnelle	BD XML native
Flexibilité	Fixe et rigide	Flexible et évolutive
Typage	Fortement typée	-Fortement typée en la présence d'un schéma XML. -Non typée en l'absence d'un schéma XSD.
Présence d'un schéma	Toujours présent	Peut ne pas être présent
Régularité de la structure	Très régulière	Peu régulière (car XML est semi-structuré) et parfois franchement irrégulière
Type de la structure	Tabulaire	Arborescente ou sous forme de graphe
Type de stockage	Eclate le document original (transformation du document original)	Conserve le document tel quel
Intégrité référentielle	Les clés étrangères doivent pointer sur des clés primaires valides	Validité des pointeurs vers des documents externes, vers des fragments de documents et vers des éléments externes ou internes
Valeurs nulles et index	Existants	Existants
Langages de requêtes	Propres aux BD relationnelles	Propres aux BD XML natives
Sécurité, transactions, triggers, accès multi-utilisateurs, requêtes sur plusieurs documents	Existants et performants	Existants mais restent encore à développer et à améliorer

Après avoir analysé ce tableau nous pouvons conclure que :

-Contrairement aux BD relationnelles, les BD XML natives sont très flexibles, ce qui les rend évolutives et plus riches, ceci est dû au fait que le langage XML est souple et flexible.

-La notion de typage est toujours présente dans les BD relationnelles mais elle peut être absente dans les BD XML natives (car ceci dépend de la présence du schéma associé au document XML).

-Une BD relationnelle peut contenir plusieurs tables donc l'interrogation de cette dernière peut nécessiter plusieurs jointures, par contre une BD XML native représentant le même document nécessitera un seul accès et pas de jointures, donc un accès plus rapide.

-Pour stocker l'information, une BD relationnelle éclate le document original en plusieurs tables, ce dernier perd son aspect d'origine, contrairement aux BD XML natives qui garde le document dans son état initial, ceci s'avère important dans le cas où le document doit servir de preuve légale par exemple en cas de litige.

-Vu que les BD XML natives sont encore à leur début, les aspects de sécurité, de transactions, de triggers…sont encore à performer et ils sont toujours en cours de recherche dans les grandes universités, par contre ses aspects là sont à leurs top dans les BD relationnelles et c'est ce qui a d'ailleurs fait leur force et leur puissance jusqu'à présent.

Chapitre4: Problématique et Solution Proposée

4.1 Problématique

Comme nous l'avons décrit dans le chapitre 1, parmi les étapes de construction d'un Data Warehouse, l'étape d'intégration est une étape cruciale, elle peut s'avérer très complexe, longue, fastidieuse et pose souvent des problèmes de qualification sémantique de données hétérogènes à intégrer. Par expérience, elle représente 60 à 90 % de la charge d'un projet de construction du Data Warehouse.

Face à l'hétérogénéité des sources de données et à la difficulté de leur intégration au sein d'un Data Warehouse, plusieurs chercheurs ont conçu et mis en œuvre des applications basées sur un format pivot d'échange qui est le langage XML [13], [14] (Figure1). En effet, la solution proposée à ce problème dans [13] et [14] est de se baser sur un format unifié d'échange de données, il s'agit en fait, de construire une vue XML et son schéma XSD pour chaque source de données externe au Data Warehouse et aussi pour représenter l'information contenue dans le Data Warehouse. Ainsi, toutes les données qu'elles soient internes ou externes à l'entrepôt seront représentées sous le même format [50], [51].

A ce niveau, l'étape d'intégration devient plus simple à réaliser, car il s'agit de comparer le schéma des données externes au Data Warehouse avec celui de ses données internes. Le but de cette comparaison est le suivant : Si les deux schémas XSD (celui du Data Warehouse et de la source de données) sont équivalents alors la source de données est intégrable (Figure2).

Cette idée, qui a été présenté dans [1] est insuffisante car certes elle présente une méthode basée sur la vérification de la sémantique et le contenu des éléments mais elle n'est pas EXHAUSTIVE. La méthode que nous avons proposée et qui est décrite ci-dessus est une méthode exhaustive et puissante : elle traite tous les cas possibles qui peuvent se présenter dans un schéma XSD, ainsi elle analyse cas par cas tous les cas de conflits qui peuvent se poser entre deux schémas XSD distincts quelque soit leurs degrés de complexité, quelque soit le degré d'imbrication de leurs éléments, quelque soit l'ordre et l'occurrence de leurs apparitions et quelque soit les différences de leurs types (Figure3).

Il est très important et intéressant de comparer les deux schémas car un schéma XSD contient la structure exacte (grâce aux types de données) et la sémantique (grâce aux noms d'éléments et d'attributs) d'un document XML, ce dernier ne comporte que le contenu informationnel brut du document.

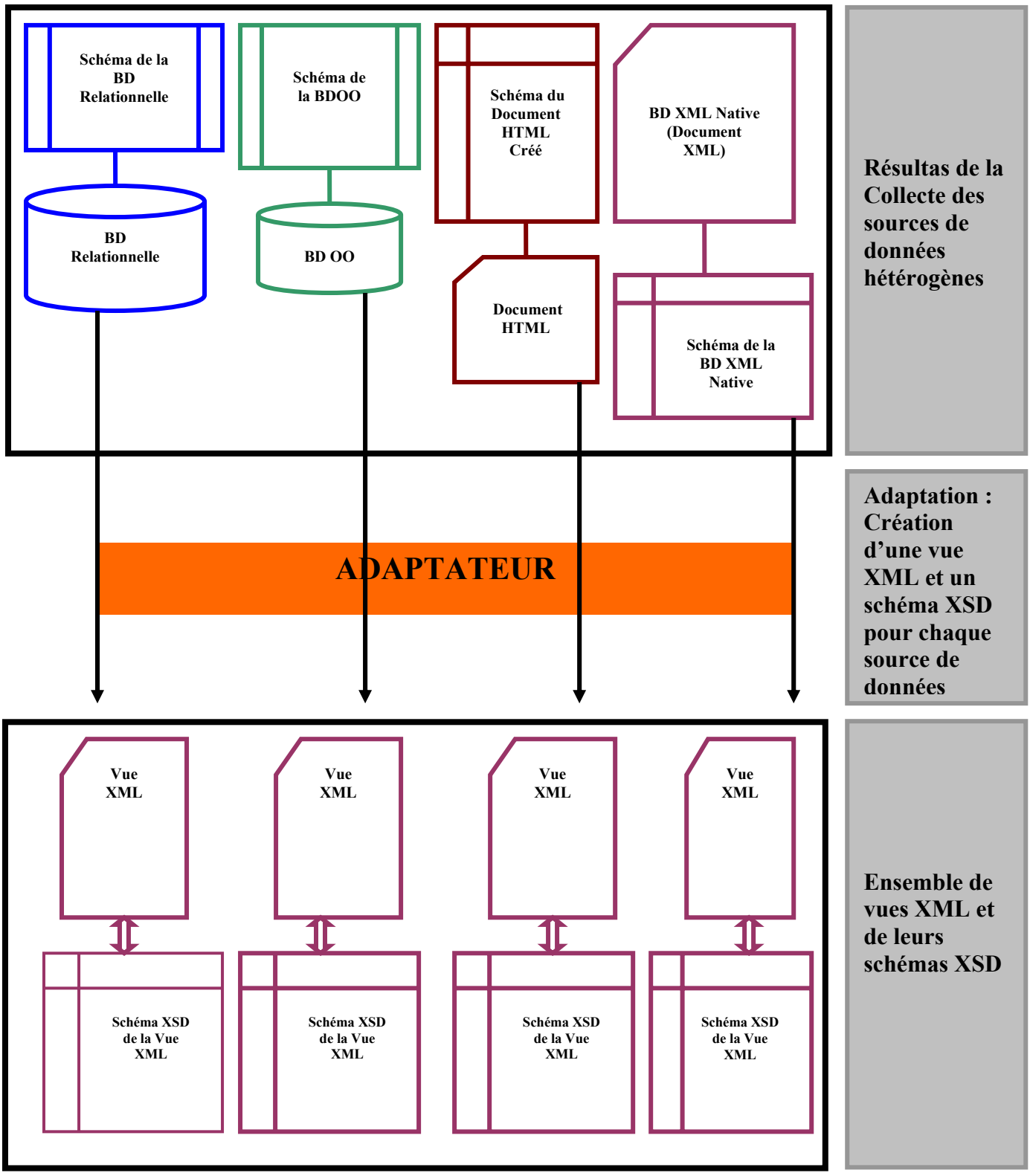


Figure 1: Unification des sources de données hétérogènes

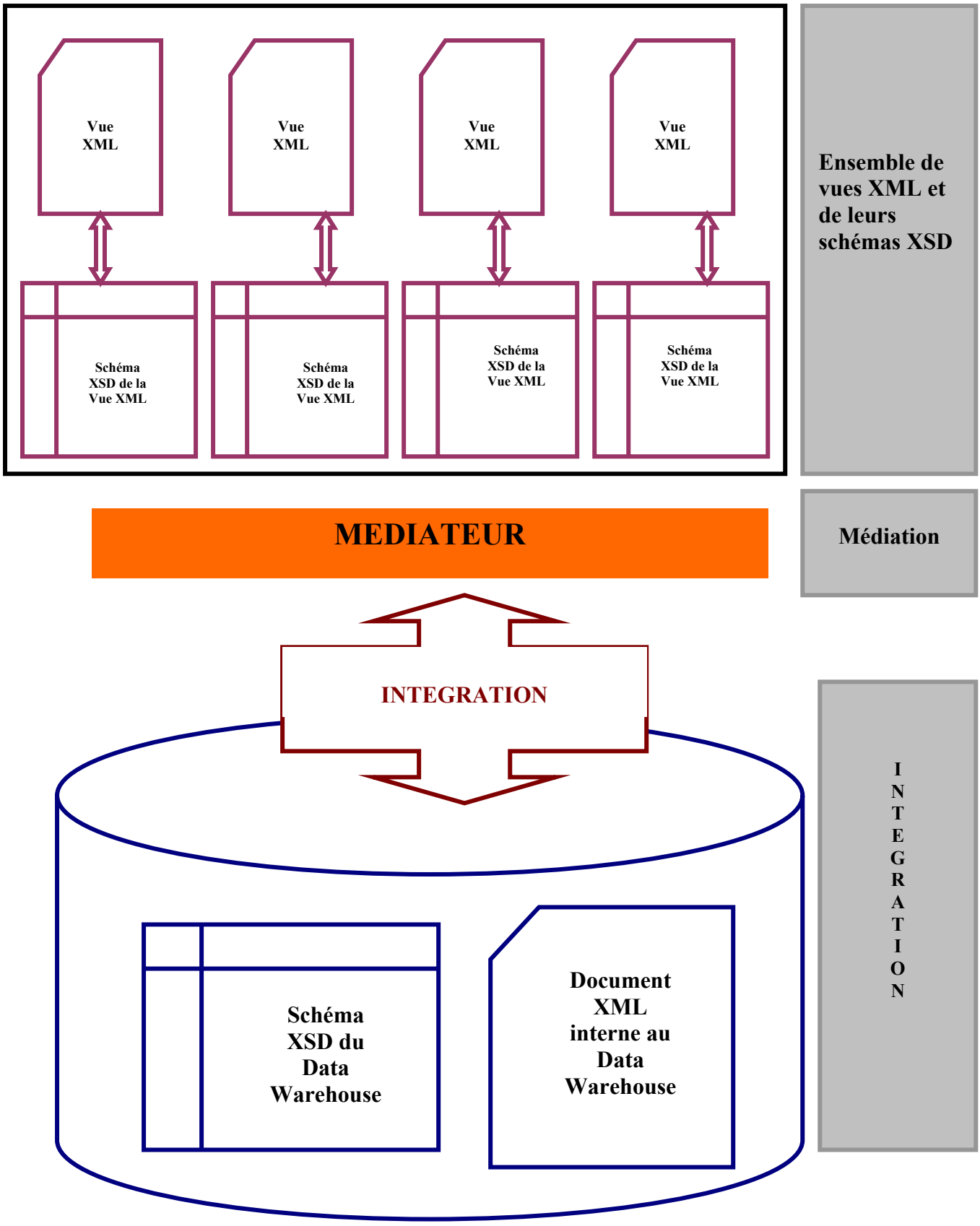


Figure 2 : Architecture générale de l'intégration au sein du Data Warehouse

4.2 Solution proposée et architecture générale du système

La solution que nous avons proposée se base principalement sur la comparaison de deux schémas XSD (cible et externe). Pour cela, nous avons besoin d'analyser et de comparer les deux éléments racines des deux schémas, et comme un élément dans un schéma est caractérisé par son nom et son type : nous comparons d'abords les noms des éléments provenant des deux schémas et ensuite leurs types et leurs attributs (La Figure3 montre l'architecture générale de la solution proposée). La comparaison de deux types d'éléments différents implique nécessairement la comparaison de leurs fils respectifs et leurs attributs : leurs noms et leurs types, cette dernière nécessite à son tour la comparaison des fils de ces fils et leurs attributs: leurs noms et leurs types … et ainsi de suite.

Le type d'un élément peut être atomique, simple ou complexe, il peut porter une ou plusieurs facettes (dans le cas où il est simple ou atomique), il peut aussi avoir certaines contraintes sur son nombre d'apparition, il peut également avoir des contraintes sur l'apparition de ses attributs, ces derniers peuvent avoir des contraintes sur leurs types. Si le type d'un élément est complexe, il peut avoir des contraintes sur l'imbrication et l'ordre d'apparition des ses fils.

Toutes ces différences représentent des conflits pour l'intégration, et notre méthode les traite cas par cas afin de pouvoir intégrer que les données correspondantes au schéma du Data Warehouse.

La comparaison des noms d'éléments (et aussi des noms des attributs) est une étape importante car dans un document XML, l'utilisateur donne à ses éléments des noms parlants qu'il a lui-même choisi, ces noms correspondent exactement au contenu et à la sémantique de l'instance XML. De plus, ce langage permet de mettre en valeur la sémantique des documents et permet à l'utilisateur de créer son propre jeu de balise. La comparaison des noms d'éléments se fait grâce à un dictionnaire électronique appelé **WordNet**.

Si deux éléments (et aussi deux attributs) ont des noms et des types compatibles alors le schéma externe peut être intégré car il correspond parfaitement à notre schéma cible.

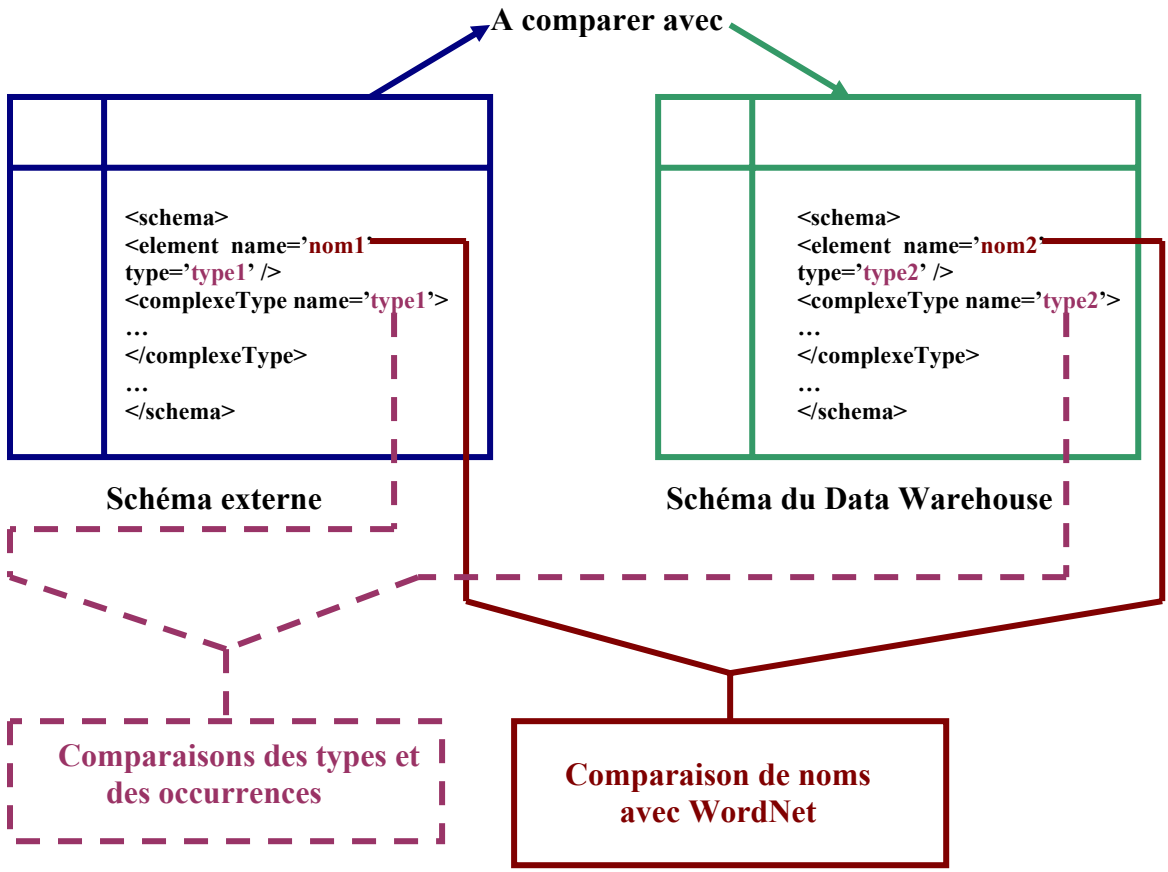


Figure 3 : Architecture générale de la solution proposée : Intégration basée sur la comparaison des schémas XSD

4.3 La vérification sémantique avant l'intégration : Traitement des cas de conflits

La solution que nous avons proposée se base sur la comparaison de deux éléments racines des schémas du Data Warehouse et celui de la source de données externe, et comme cité précédemment, cette comparaison s'applique aux noms d'éléments, leurs types, leurs occurrences et leurs attributs, ce traitement peut générer plusieurs cas de conflit que nous allons traiter en détail dans ce qui suit. Toute cette étude se fait dans le cadre de la vérification sémantique des deux schémas avant l'intégration au sein du Data Warehouse.

Les cas de conflits qui peuvent se présenter sont :

- Cas de conflit des noms d'éléments.
- Cas de conflits des occurrences des éléments.
- Cas de conflits des types d'éléments.
- Cas de conflits des attributs (noms et types) portés par les éléments.
- Cas de conflit des facettes appliquées aux types simples d'éléments ou d'attributs.

4.3.1 Cas de conflit des noms d'éléments

Deux noms d'éléments sont dits équivalents si le résultat de la fonction **WordNet** appliquée à ces deux éléments retourne TRUE. La fonction **WordNet** est tout simplement un dictionnaire électronique qui a en entrée deux mots et qui donne en sortie un résultat booléen : TRUE ou FALSE.

Deux éléments peuvent avoir exactement les mêmes noms ou des noms équivalents mais avoir deux types complètement différents et c'est pour cela que notre système va au-delà de ce traitement : il traite aussi les types des éléments.

Si le résultat de la fonction **WordNet** retourne FALSE notre système vérifie aussi leurs types respectifs, si ces derniers sont compatibles alors notre application génère un message (une sorte d'alerte) indiquant à l'utilisateur de vérifier les noms de ses éléments avant la phase d'intégration.

4.3.2 Cas de conflit des occurrences des éléments

Ce module traite les occurrences des éléments selon les valeurs des attributs minOccurs et maxOccurs (attributs indiquant le nombre d'apparition d'un élément), pour cela nous distinguons plusieurs cas :

Si l'élément du schéma cible ne doit pas apparaître alors l'élément du schéma externe ne doit par apparaître non plus. Et dans le cas où le premier élément doit apparaître alors le deuxième élément doit apparaître aussi.
De plus, si le premier élément apparaît un nombre fixe de fois alors le deuxième élément doit apparaître exactement ce nombre de fois.
Enfin, si le nombre d'occurrence du premier élément appartient à une plage de valeurs alors le deuxième élément doit apparaître un nombre de fois inclus dans cette plage de valeurs.

Exemple :

L'élément interne au DW :
<element name=*'code'* type='integer' minOccurs='0' maxOccurs='unbounded'/>

Dans ce cas, l'élément ***code*** est un élément de type atomique ***integer*** qui doit peut apparaître 0, 1 ou plusieurs fois.

L'élément externe à intégrer dans le DW :
<element name=*'code'* type= '*integer'* minOccurs= '0' maxOccurrs='4' />

Dans ce cas, l'élément ***code*** est d'un type atomique ***integer*** qui peut apparaître au plus 4 fois.

L'intégration est possible, car nous avons deux éléments de mêmes noms et de mêmes types. De plus, leurs nombres d'apparition ne génère pas de conflit, car si un élément apparaît plusieurs fois (le nombre maximum d'apparition n'est pas déterminé) cela veut dire qu'il peut apparaître au plus 4 fois.

Si nous inversons les deux éléments, l'intégration devient impossible car nous ne savons pas à priori combien de fois peut apparaître un élément quand il a nombre indéterminé

d'occurrences et dans l'autre élément le nombre maximum d'apparition est de 4. En d'autres termes, l'élément externe peut apparaître 5 fois ce qui génère un conflit d'intégration.

4.3.3 Cas de conflit des types d'éléments

Cette étape traite les types d'éléments possibles dans un schéma XSD, à savoir : les types atomiques, simples et complexes (Figure5).

Figure 5 : Les types de données d'un élément dans un schéma

L'hétérogénéité des types représente un conflit pour l'intégration, car même si deux types sont de même nature (tous les deux atomiques, simples ou complexes), ils peuvent être complètement différents. Si les deux types sont de natures différentes (par exemple l'un est simple et l'autre est complexe) alors il y a nécessairement un conflit.

Notre système traite tous les cas de conflits possibles, cette méthode est exhaustive car elle traite tous les cas de conflits qui peuvent se présenter et tous les fils possibles de chaque type.

Un conflit ne veut pas dire nécessairement que l'intégration est impossible car –nous le verrons plus en détail dans ce qui suivra- un type simple et un type complexe peuvent être compatibles car leur contenu l'est aussi. Ce point montre que notre application est assez souple et flexible vu qu'elle ne retourne pas un résultat négatif dès qu'elle trouve un conflit de types : elle traite les fils de chacun des types pour examiner leurs structures respectives.

4.3.3.1 Rappel des définitions des types d'un élément

Nous allons rappeler dans ce qui suit la définition de chacun des types que peut porter un élément :

1) Atomique : C'est un type appartenant à la bibliothèque du langage des schémas XSD. Ces types sont fixes et ont un nombre limité (voir tableau page 36 du Chapitre 2).

2) Simple : C'est un type qui peut être soit une restriction d'un type simple en lui appliquant une ou plusieurs facettes, soit une liste de types simple (le même type simple), soit une union de types simples (plusieurs types simples).

3) Complexe : C'est un type contenant un ou plusieurs fils et éventuellement des attributs. Un type complexe peut avoir comme contenu :

a) **Un simpleContent** : Il est utilisé lorsque l'on veut transformer un type simple en un type complexe en lui rajoutant des attributs (cette méthode est appelé : extension) ou –en plus des attributs- lui appliquer une ou plusieurs facettes (cette méthode est appelé : restriction).

b) Un complexContent : Il est utilisé lorsque l'on veut ajouter (extension) ou éliminer (restriction) des éléments ou leur nombre d'apparition dans un type qui est complexe à la base et cela en utilisant les éléments : séquence, all, choice et group, ces éléments seront détaillés ci-dessous.

c) Une séquence : Elle est utilisée lorsque l'on veut imposer l'apparition de plusieurs éléments dans un ordre bien précis.

d) Un all : Il est utilisé lorsque l'on veut qu'un ou plusieurs éléments apparaissent ou non et dans n'importe quel ordre.

e) Un choice : Il est utilisé lorsque l'on veut qu'un élément (ou un groupe d'éléments) apparaisse au choix parmi tous les fils possible de choice.

f) Un group : Il est utilisé lorsque l'on veut décrire un élément de type complexe ayant comme fils un mélange d'un ou plusieurs séquence, choice et all (Figure6) [30].

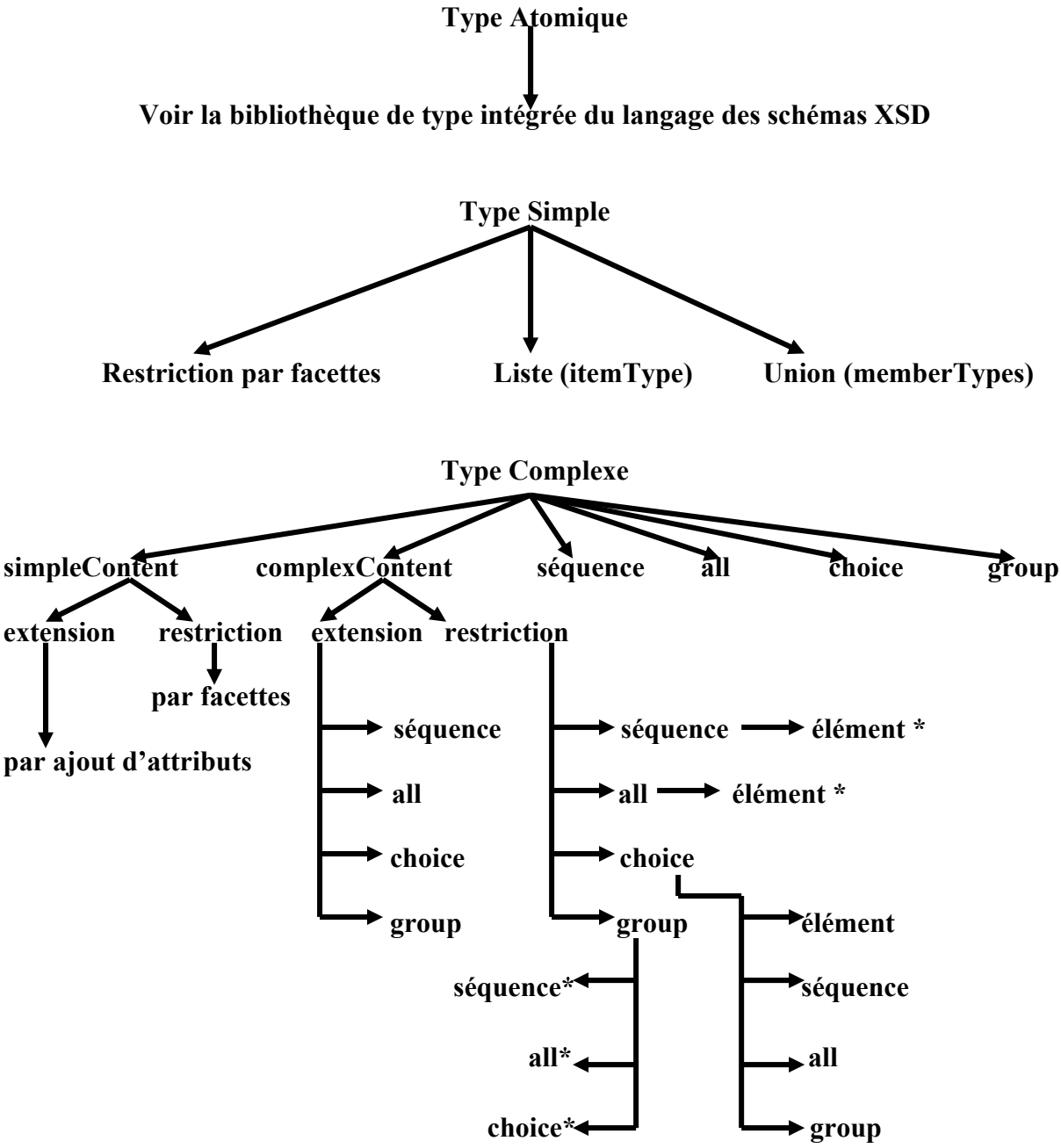


Figure 6: La description des types Atomique, Simple et Complexe

Nous allons à présent illustrer comment notre système traite les cas de conflit entre deux éléments de mêmes types ou de types différents en donnant à chaque fois un exemple montrant cette solution.

4.3.3.2 Les deux types sont de même nature : ATOMIQUE

Dans ce cas, si les deux éléments ont exactement les mêmes types (ou des types équivalents) et les mêmes occurrences (ou des occurrences équivalentes –voir le traitement des occurrences plus haut-) alors ils sont compatibles sinon leurs types sont différents et l'élément du schéma externe est non-intégrable.

Dans un premier temps nous allons donner un exemple de deux éléments de même type atomique.

Exemple :

Prenons ce premier cas :

L'élément interne au DW :
<element name=*'enseignant'* type=*'string'*/>

L'élément externe à intégrer dans le DW :
<element name=*'professeur'* type=*'string'*/>

Dans ce cas, l'intégration est possible car nous avons deux éléments de mêmes types et portant des noms équivalents.

Prenons ce deuxième cas :

L'élément interne au DW :
<element name=*'enseignant'* type=*'string'*/>

L'élément externe à intégrer dans le DW :
<element name=*'voiture'* type=*'string'*/>

Dans ce cas, l'intégration est impossible car nous avons deux éléments de mêmes types mais portant des noms tout à fait différents.

Dans ce qui suit, nous allons montrer dans quel cas, deux types atomiques sont différents (de noms) mais équivalents.

Si les deux types sont atomiques, ils peuvent être différents mais représentant exactement la même valeur, par exemple : un élément de type ***integer*** de valeur '1' est exactement le même qu'un élément portant le même nom de type ***decimal*** de valeur '1.0'.

Afin de résoudre les problèmes d'hétérogénéité entre deux types atomiques, nous les avons réparti en quatre catégories principales, les significations de ces types sont décrites dans le

chapitre 2 à la page 36. Dans chaque catégorie, un exemple d'intégration est donné afin de montrer les cas qui peuvent –ou non- poser problème.

Catégorie 1 : Elle regroupe tous les types atomiques de type chaîne de caractères, à savoir : String, Name, Qname, Ncname, Notation, normalizedString, token, language, ID, IDS, IDREF, IDREFS, ENTITY, ENTITIES, NMTOKEN, NMTOKENS et anyURI.

Tous les types de la Catégorie 1 sont à la base des types String (ils sont tous dérivés du type String), par conséquent équivalent au type String. Par contre, deux autres types de cette catégorie sont compatibles si le premier type respecte les conditions syntaxiques du deuxième type.

Dans un cas de conflit entre deux types de la même catégorie 1, il suffit de vérifier que ces deux types ont les mêmes caractéristiques, par exemple, les deux types Name (type de l'élément interne au DW) et String (type de l'élément externe au DW) sont exactement les mêmes dans le seul cas où l'élément de type String est un nom XML (d'après les caractéristiques d'un nom XML vu dans le chapitre 2).
Par contre si l'élément interne au DW est de type String et que l'élément externe au DW est de type Name, l'intégration est toujours possible car un type Name est à la base un type String.

Catégorie 2 : Elle regroupe tous les types atomiques de type numérique, à savoir : Decimal, Float, Double, Integer, NonPositiveInteger, NegativeInteger, Long, Int, Short, Byte, NonNegativeInteger, PositiveInteger, UnsignedLong, UnsignedInt, UnsignedShort, UnsignedByte.

La Catégorie 2 a deux sous-catégories :

- **Catégorie a :** Elle contient des types décimaux : Decimal, Float, Double. La différence entre ces types est la plage de valeurs qu'ils comportent : un type Double contient un type Float, et un type Float contient un type Decimal. De ce fait, par exemple, un type Float est un type Decimal mais l'inverse n'est pas vrai.

- **Catégorie b :** Elle contient des types entier : Integer, NonPositiveInteger, NegativeInteger, Long, Int, Short, Byte, NonNegativeInteger, PositiveInteger, UnsignedLong, UnsignedInt, UnsignedShort, UnsignedByte. La différence entre ces types est la plage des valeurs qu'ils comportent, leurs signes (positif ou négatif), et la valeur 0. Par exemple, un type Integer est un type NonPositiveInteger que si le type Integer est toujours positif ou nul mais l'inverse est toujours vrai.

Dans un cas de conflit entre un type Decimal (type de l'élément interne au DW appartenant à la Catégorie a) et un type Integer (type de l'élément externe au DW appartenant à la Catégorie b), l'intégration est tout à fait possible car un type Integer est un type Decimal.
Par contre, dans le cas contraire, un type Decimal ne peut être équivalent à un type Integer que si il porte la valeur 0 (ou pluiseurs 0) après la virgule. Cette condition est vérifiable en utilisant des facettes (elles seront décrites plus loin).

Catégorie 3 : Elle regroupe tous les types indiquant des valeurs dans le temps, à savoir : dateTime, Duration, Time, Date, gYearMonth, gYear, gMonthDay, gDay, gMonth, TimeInstant, TimeDuration et ReccuringInstant.

Tous les types de la Catégorie 3 représente une valeur dans le temps mais avec une syntaxe bien précise.

Dans un cas de conflit entre un type gDay (type de l'élément externe au DW ayant la syntaxe AAAA-MM-JJ) et un type Date (type de l'élément interne au DW ayant la syntaxe AAAA-MM-JJ+HH :MM), l'intégration est possible car le type de l'élément externe indique une date tel que c'est le cas dans l'élément interne, la seule différence, c'est la précision de l'heure.
Par contre, dans le cas inverse, l'intégration est impossible car la précision de l'heure est requise dans le type de l'élément du schéma du DW.

Catégorie 4 : Elle contient un seul type qui est le type Boolean, dans ce cas, l'intégration n'est possible que si les deux types sont Boolean.

Pour conclure, deux types peuvent être équivalent s'ils appartiennent à la même catégorie tout en ayant des syntaxes compatibles tel que c'est décrit plus haut.

Dans tout ce qui suivra, nous allons qualifier deux types (atomique et un autre) de même type qu'il soient exactement les mêmes ou qu'ils soient équivalents.

4.3.3.3 Les deux types sont de natures différentes : ATOMIQUE et SIMPLE

Ce module traite le cas de conflit entre un type atomique et un type simple, pour cela nous allons traiter le type atomique avec tous les cas de figure d'un type simple à savoir : une restriction, une liste et une union.

Dans le cas d'une restriction, si le type atomique est le même que le type de base de la restriction (car si avant la restriction les types sont déjà différents, il est impossible que ces types deviennent équivalents après la restriction) et que les deux éléments ont des occurrences équivalentes et que le type atomique n'est pas en conflit avec les facettes appliquées au type simple (car l'application d'une facette réduit les valeurs du type simple et ce dernier peut ne plus être compatible avec le type atomique, nous détaillerons cette partie dans le traitement des facettes plus loin) alors les deux types sont compatibles.
Le traitement d'une liste est le même traitement que celui d'une restriction sauf que la condition de l'occurrence n'est pas nécessaire à vérifier car nous ne savons pas à priori le nombre de fois que va apparaître l'élément de type liste, de plus l'élément de type atomique doit avoir une occurrence non-nulle car la liste comporte au moins un élément.
Le traitement d'une union est le même que celui d'une liste dans le seul cas où l'union ne se compose que d'un seul memberType.

Exemple :

L'élément interne au DW :

<element name=*'âge'* type='type_age' />
<*simpleType* name='type_age'>
<*list* itemType=*'integer'*/>
</simpleType>

Dans ce cas, l'élément ***âge*** est une liste (un type simple) de plusieurs age dont le type est ***integer***.

L'élément externe à intégrer dans le DW :

<element name=*'âge'* type= '*integer'* minOccurs= '0' maxOccurrs='4' />
Dans ce cas, l'élément ***âge*** est d'un type ***atomique*** qui peut apparaître au plus 4 fois.

L'intégration est possible, car nous avons deux éléments de mêmes noms et avec des types compatibles (apparition d'un integer plusieurs fois).

Si nous inversons les deux éléments, l'intégration devient impossible car nous ne savons pas à priori combien de fois peut apparaître un élément dans une liste et dans l'autre élément le nombre maximum d'apparition est de 4.

4.3.3.4 Les deux types sont de même nature : SIMPLE

Ce module traite le cas de conflit entre deux types simples (restriction, liste et union).
Dans le cas où les deux types sont de même nature : si c'est des listes avec les mêmes itemTypes alors les deux éléments sont compatibles, si c'est des unions alors elles doivent avoir les mêmes memberTypes et si c'est des restrictions alors elles doivent avoir les même types de base, des occurrences équivalentes et compatibles après l'application de facettes.
Par ailleurs, dans le cas où les deux types simples ne sont pas de même nature alors : la liste et l'union ne sont compatibles que si elles ont les même itemTypes/memberTypes, la liste/l'union et la restriction ne sont compatibles que si elles ont le même type de base et l'application des facettes ne génère pas de conflit avec le type des itemType/memberTypes.

Exemple :

L'élément interne au DW :

```
<element name='moyenne' type='type_moyenne' />
<simpleType name='type_moyenne'>
<list itemType='moyenne_minimum'/>
</simpleType>
<simpleType name= 'moyenne_minimum' >
<restriction base='real'>
<minInclusive value='10' />
</restriction>
</simpleType>
```

L'élément ***moyenne*** est une liste de real supérieur ou égal à 10.

L'élément externe à intégrer dans le DW :

```
<element name='moyenne' type='type_moyenne' />
<simpleType name='type_moyenne'>
<list itemType='moyenne_minimum'/>
</simpleType>
<simpleType name= 'moyenne_minimum' >
```

<*restriction* base='real'>
<*minExclusive* value='10' />
</restriction>
</simpleType>

L'élément ***moyenne*** est une liste de real strictement supérieur à 10.

Ces deux éléments sont incompatibles vu qu'ils peuvent accepter des valeurs différentes selon la facette appliquée même s'ils ont le même type de base de la liste et aussi les mêmes noms, en effet, le premier élément peut inclure la valeur 10 et le second ne doit pas l'inclure.

4.3.3.5 Les deux types sont de natures différentes : ATOMIQUE et COMPLEXE

Ce module traite le cas de conflit entre un type atomique et un type complexe, pour cela nous devons traiter ce type atomique avec toutes les valeurs que peut prendre un type complexe.
Si le type complexe est un simpleContent (un type simple étendu avec des attributs) alors les deux éléments (atomique et complexe) doivent être compatibles et si la vérification des attributs est nécessaire alors les deux éléments sont incompatibles car le premier n'en porte pas. Si le type complexe est un complexContent alors le type atomique doit être compatible avec une des valeurs possibles du complexContent : si c'est une séquence alors tous ses fils doivent être du même type que l'élément du type atomique et ce dernier doit avoir les mêmes occurrences que le nombre de fils de la séquence. Si c'est un all alors il ne doit avoir qu'un seul fils qui doit être compatible avec le type atomique. Le traitement de choice avec un élément ne se fait que lorsque choice n'a qu'un seul fils qui peut apparaître et la comparaison est à ce moment là possible, car si choice a plusieurs fils, la comparaison est impossible du fait que nous ne pouvons pas savoir à priori quel est le fils de choice qui va apparaître pour pouvoir le comparer. Un type atomique n'est compatible avec un choice que si ce dernier a un fils compatible avec le type atomique. Enfin, un type atomique n'est compatible avec un group que si ce dernier n'a qu'un seul fils de nature compatible avec le type atomique.

Exemple :

L'élément interne au DW :

<element name=*'prénoms'* type=*'string'* minOccurs='2' maxOccurs='2' />

L'élément prénoms est un élément de type ***atomique string*** qui doit apparaître 2 fois.

L'élément externe à intégrer dans le DW :

<element name=*'prénoms'* type='type_personne'/>
<*complexType* name= 'type_personne'>
<*sequence*>
<element name='premier_prénom' type=*'string'* />
<element name='deuxième_prénom' type=*'string'* />
< /sequence>
</complexType>

L'élément prénoms est un élément de type ***complexe*** qui est en fait une ***séquence*** de exactement deux éléments de type ***string***.

Ces deux éléments sont équivalents même s'ils ont des types différents car ils permettent de générer deux instances XML compatibles.

4.3.3.6 Les deux types sont de natures différentes : SIMPLE et COMPLEXE

Ce module traite le cas de conflit entre un type simple et un type complexe. Afin d'aboutir à un traitement complet, nous allons détailler tous les cas de figure possibles entre un type simple (qui peut être une restriction, une liste ou une union) et un type complexe (qui peut être un simpleContent, complexContent, une séquence, un all, un choice ou un group).

Si le type simple est une restriction, une liste ou une union et le type complexe est un simpleContent alors ces deux types ne sont compatibles que si ils ont le même type même après l'application de facettes, mais si la vérification des attributs est nécessaire, les deux types sont toujours incompatibles car le type simple n'en porte pas. Par ailleurs, si le type complexe est un complexContent alors le type simple doit être compatible avec tous les fils du complexContent (séquence, choice, all et group).

Si le type simple est une liste (respectivement une union) et le type complexe est un complexContent alors les deux types sont compatibles que si les éléments présents dans la séquence (l'ordre de l'apparition des fils doit être respecté), all, choice (un seul fils de choice doit apparaître) ou group sont tous compatibles avec le type de l'itemType de la liste (respectivement le type des memberTypes de l'union).

Exemple :

L'élément interne au DW :

```
<element name='numéro_téléphone' type='num' />
<complexType name='num'>
<all>
<element name='téléphone1' type='lenght_tel1' minOccurs='0' maxOccurs='unbounded' />
<element name='téléphone2' type='lenght_tel2' minOccurs='0' maxOccurs='unbounded' />
<simpleType name='lenght_tel1'>
<list itemType='long_tel'/>
</simpleType>
<simpleType name= 'long_tel' >
<restriction base='integer'>
<minLength value='9' />
</restriction>
</simpleType>
<simpleType name='lenght_tel2'>
<list itemType='long_tel'/>
</simpleType>
<simpleType name= 'long_tel' >
<restriction base='integer'>
<length value='9' />
</restriction>
```

```
  </simpleType>
</all>
</complexType>
```

L'élément ***numéro_téléphone*** est un élément de type complexe ayant un fils décrivant l'apparition d'un ou plusieurs numéros de téléphone dans n'importe quel ordre de longueur supérieur ou égale à 9 chiffres.

L'élément externe à intégrer dans le DW :

```
<element name='num_téléphone' type='type_num' />
<simpleType name='type_num'>
<list itemType='longueur_tel'/>
 </simpleType>
 <simpleType name= 'longueur_tel' >
 <restriction base='integer'>
 <length value='9' />
  </restriction>
  </simpleType>
```

L'élément ***num_téléphone*** est un élément de type simple : une liste de numéro de téléphone de longueur égale à 9 chiffres.

Ces deux éléments sont équivalents même s'ils ont des types différents car ils permettent de générer deux instances XML compatibles.

4.3.3.7 Les deux types sont de même nature : COMPLEXE

Ce module traite le cas de conflit entre deux types complexes. Pour réaliser une analyse optimale, nous avons besoin de traiter toutes les combinaisons possibles entre deux types complexes (un type complexe peut être un simpleContent, un complexContent, une séquence, un all, un choice ou un group).

Si un des deux types complexes est un simpleContent alors les deux types ne sont compatibles que si ils ont le même type même après l'application de facettes et qu'ils ont des occurrences équivalentes.

Les cas de conflits les plus intéressants sont ceux qui peuvent se poser entre deux types complexes quand ils sont tous les deux des complexContent.
Si les deux types sont des séquences alors leur compatibilité nécessite que tous les éléments de deux séquences soient compatibles deux à deux et dans l'ordre de leur apparition. Et si un seul des deux types est une séquence alors le deuxième type complexe doit comporter les mêmes éléments de la séquence et dans l'ordre selon cette dernière.

Si un des deux types est un all, choice ou group alors il doit exister pour chaque élément du premier type complexe un autre élément du deuxième type complexe qui lui est compatible, car si par exemple, le premier type choice a un élément de plus que le deuxième type choice alors ces deux types sont incompatibles du fait que le premier risque de contenir un élément de plus que le second.

Exemple :

L'élément interne au DW :

<element name=*'contact'* type='type_contact'/>
<*complexeType*>
<*choice*>
<element name='tel' type='integer'/>
<element name='fax' type='integer'/>
</choice>
</complexType>

L'élément ***contact*** est un élément de type ***complexe*** se décrivant comme un ***choix*** entre l'apparition du téléphone ou du fax pour représenter ce contact.

L'élément externe à intégrer dans le DW :

<element name=*'contact'* type='type_contact'/>
<*complexeType*>
<*sequence*>
<element name='tel' type='integer'/>
<element name='fax' type='integer'/>
</sequence>
</complexType>

L'élément ***contact*** est un élément de type ***complexe*** se décrivant comme une ***séquence*** de deux éléments : téléphone et fax (dans cet ordre) pour représenter ce contact.

Ces deux éléments ne sont pas équivalent même s'ils portent le même nom car ils ont deux types différents : le premier type choisi entre l'apparition de deux éléments (un seul d'entre eux va apparaître) et le second impose l'apparition des deux éléments dans un ordre bien précis, donc dans le premier élément, nous ne savons pas quel est l'élément qui va apparaître dans l'instance XML.

4.3.4 Cas de conflit d'attributs (noms et types)

Les éléments d'un schéma peuvent porter des attributs et un attribut est toujours d'un type simple ou atomique, donc le traitement de type de deux éléments peut être suivi du traitement de leurs attributs (s'ils en portent), il suffira alors de leur appliquer le même traitement appliqué aux éléments sauf qu'ici nous traiterons que des types atomiques ou simples. De plus, si un attribut est requis dans le schéma cible alors il doit obligatoirement l'être dans le schéma externe, et si il a une valeur fixée alors cette valeur doit être la même dans le schéma externe.

4.3.5 Cas de conflit de facettes

Après avoir analysé tous les cas de conflits, il nous reste juste à détailler les conflits qui peuvent se produire après l'application d'une ou plusieurs facettes. Tout d'abord, nous savons que certaines facettes sont applicables à quelques types seulement et ne sont pas applicables à

d'autres types [29], nous citerons à titre d'exemple, la facette minInclusive qui peut être appliquée à un type numérique (entier ou réel) mais pas un type chaîne de caractère. Nous ne risquons pas de tomber dans ce problème car les documents que nous traitons sont sensés être bien formé (il existe plusieurs outils permettant de savoir si un document est bien formé ou non).

Le module qui traite les facettes s'applique à deux éléments : un élément du schéma cible et un autre du schéma externe.

4.3.5.1 Rappel sur les facettes

Nous allons, dans un premier temps, rappeler le sens de chaque facette lorsqu'elle est appliquée à un type simple :

1) length : Elle a pour but de fixer la longueur de la valeur de l'élément, cette longueur correspond à la valeur de l'attribut value porté par cette facette. Cette dernière n'est applicable qu'aux types : string, anyURI, QName, Notation, normalyzedString, token, Name, NCName, ID, IDRef, entity, entities et IDRefs (Appelons ce groupe de types : groupe A).

2) minLength (respectivement maxLentgh): Elle a pour but de fixer la longueur minimale (respectivement maximale) de la valeur de l'élément, cette longueur correspond à la valeur de l'attribut value porté par cette facette. Cette dernière est applicable aux mêmes types de la facette length.

3) pattern : Cette facette a pour but de définir une expression régulière pour la valeur d'un élément donné, c'est le cas par exemple si l'on veut éviter des adresses électroniques farfelues, nous pouvons créer un type simple avec la facette pattern portant l'attribut value à la valeur : (.) + @ + (.). Cette facette peut être appliquée à différents types simples.

4) énumération : Cette facette a pour intérêt de permettre à un élément de prendre comme valeurs (de types simples) que quelques valeurs possibles, nous citerons à titre d'exemple un élément qui va contenir comme valeur le résultat des délibérations à savoir : Admis ou Ajourné. Cette facette peut être appliquée à différents types simples.

5) totalDigits : Elle a pour but de fixer le nombre de digits qui compose la valeur de l'élément, ce nombre correspond à la valeur de l'attribut value porté par cette facette. Cette dernière n'est applicable qu'aux types: décimal, integer, nonPositiveInteger, long, nonNegativeInteger, negativeInteger, int, short, byte, unsignedLong, unsignedInt, unsignedShort, unsignedByte et positiveInteger (Appelons ce groupe de types : groupe B).

6) fractionDigits : Elle a pour but de fixer le nombre de digits après la virgule dans la valeur de l'élément et elle s'applique aux mêmes types de la facette totalDigits (les types du groupe B).

7) minInclusive (respectivement maxInclusive) : Elle a pour but de fixer la plus petite (respectivement grande) valeur que peut avoir la valeur d'un élément de type appartenant au groupe B (cette valeur étant incluse), elle est aussi applicable aux types : float, double, duration, dateTime, time, date, gYearMonth, gYear, gMonthDay, gDay et gMonth (Appelons ce groupe de types : groupe C et groupe D est le groupe qui contient les valeurs du groupe B et C).

8) minExclusive (respectivement maxExclusive) : Elle a pour fixer la plus petite (respectivement grande) valeur que peut avoir la valeur d'un élément de type appartenant au groupe D (cette valeur n'étant pas incluse) [29].

4.3.5.2 Comparaison de types après application des facettes

Pour illustrer le traitement des facettes, nous allons énumérer toutes les facettes appartenant au langage des schémas XSD en soulignant à chaque fois les cas de conflits qui peuvent se produire :

1) length : Cette facette ne génère pas de conflit pour l'intégration dans le cas où la facette appliquée à l'élément externe a une valeur égale à celle indiquée par la facette length.

2) minLength : Cette facette ne génère pas d'incompatibilité pour l'intégration dans le cas où la facette appliquée à l'élément externe a une valeur supérieure à celle indiquée par la facette minLength.

3) maxLength : Cette facette ne génère pas de non-concordance pour l'intégration dans le cas où la facette appliquée à l'élément externe a une valeur inférieure à celle indiquée par la facette maxLength.

4) pattern : Cette facette ne génère pas de conflit pour l'intégration dans le cas où la facette appliquée à l'élément externe a une syntaxe identique à celle indiquée par la facette pattern.

5) énumération : Cette facette ne génère pas d'incompatibilité pour l'intégration dans le cas où la facette appliquée à l'élément externe a des valeurs identiques à celles indiquées par la facette énumération.

6) totalDigits : Cette facette ne génère pas de non-concordance pour l'intégration dans le cas où la facette appliquée à l'élément externe a une valeurs ayant le même nombre de digits que celui indiqué par la facette totalDigits.

7) fractionDigits : Cette facette ne génère pas de conflit pour l'intégration dans le cas où la facette appliquée à l'élément externe a une valeurs ayant le même nombre de digits après la virgule que celui indiqué par la facette totalDigits.

8) minInclusive (respectivement minExclusive): Cette facette ne génère pas d'incompatibilité pour l'intégration dans le cas où la facette appliquée à l'élément externe a une valeur supérieure ou égale (respectivement supérieure) à celle indiquée par la facette minInclusive (respectivement minExclusive) .

9) maxInclusive (respectivement maxExclusive): Cette facette ne génère pas de non-concordance pour l'intégration dans le cas où la facette appliquée à l'élément externe a une valeur inférieure ou égale (respectivement inférieure) à celle indiquée par la facette maxInclusive (respectivement maxExclusive).

Conclusion générale

Conclusion générale

Le travail de notre thèse s'inscrit dans le cadre de l'étude de la conception d'un entrepôt de données et principalement d'une phase cruciale dans la construction d'un Data Warehouse qui n'est rien d'autre que la phase d'intégration. Plusieurs travaux de recherche ont été dirigés pour proposer une solution aux problèmes d'intégration. Les chercheurs ont proposé l'utilisation d'un format pivot unique d'échange de données, à cet effet ils préconisent l'utilisation de XML et des schémas XSD du fait qu'ils contiennent des informations complètes et précises des sources de données.

Nous avons aussi étudié les concepts du langage XML et tout ce qu'il offre comme outils pour la description de documents structurés : les schémas XSD, tout en insistant sur leurs avantages quant à une bonne définition d'un modèle de documents contrôlables (les aspects de cohérence et d'intégrité, contrôle de données, typage, plage d'occurrences…etc.).

L'objectif de notre thèse a été de proposer une méthode complète et simple de comparaison entre le schéma du Data Warehouse et le/les schémas provenant de sources externes pouvant être incompatibles avec le schéma de l'entrepôt de données, dans le but d'assurer une intégration de données compatibles avec le schéma du Data Warehouse et cela d'une manière sûre et automatique. La méthode proposée dans ce travail traite tous les cas de conflits qui peuvent se poser entre deux schémas XSD distincts. Comme un élément d'un schéma se décrit par son nom et son type, notre méthode se base principalement sur la comparaison des noms et types des deux éléments racines des deux schémas. La comparaison des noms est assez simple car elle se base sur l'utilisation d'un dictionnaire électronique mais le type d'un élément peut être simple ou complexe. Pour traiter un type d'élément, nous nous intéressons a ses fils qui décrivent sa structure : éléments fils en séquence, un choix d'éléments fils, un groupe d'éléments, de séquence et de choix … etc. Et tout traitement d'un élément revient au traitement de ses petits fils : leurs noms et leurs types aussi. Il existe plusieurs descriptions de fils d'éléments pour définir un type complexe, tout dépend de la structure que l'on veut avoir dans nos documents XML. La méthode que nous avons proposée traite les types d'éléments quelque soit leur degré de complexité car un schéma permet de décrire toute sorte de contrainte tout en obéissant à une syntaxe définie par ses concepteurs (le W3C).

Perspectives

Une validation de l'approche est en cours de réalisation, dans l'attente de sa mise en œuvre. D'autre part, il serait intéressant de proposer une solution au problème de rafraîchissement de données car quand les données sont transformées et intégrées dans le Data Warehouse, leurs sources d'origine peuvent changer et évoluer dans le temps. Le problème des mises à jour des données et leur synchronisation peut devenir une difficulté majeure si elles ne sont pas prises en charge correctement.
Comme autres perspectives, ils est également intéressant de résoudre les problèmes des: transactions, verrouillages, sécurité et accès concurrentiels qui se posent aux niveaux des bases de données XML natives.

Bibliographie

Bibliographie

[1] A.Boukottaya, C.Vanoirbeek, F.Paganelli, O.Abou Khaled. *Automating XML documents Transformations: A conceptual modelling based approach.* In Proceedings of the first Asian-Pacific conference on Conceptual modelling, ACM International Conference Proceeding Series. Dunedin, New Zealand 2004.

[2] O.Teste. *Modélisation et manipulation d'entrepôts de données complexes et historisés.* Thèse de doctorat. Institut de recherche en informatique de TOULOUSE. Publication : Laboratoire IRIT – Pôle SIG 2000.

[3] R. Kimball, L. Reeves, M. Ross, W. Thornthwaite. *Concevoir et déployer un Data Warehouse: Guide de conduite de projet*. Edition Eyrolls, 2000.

[4] M. Jambu. *Introduction au Data Mining : Analyse intelligente de données*. Edition Eyrolles, 1997.

[5] J.M Franco, Sandrine. *Piloter l'entreprise grâce au Data Warehouse*. Edition Eyrolles, 2000.

[6] J.M Gouarné. *Le projet décisionnel : Enjeux, modèles, architecture du Data Warehouse*. Edition Eyrolles, Novembre 1997.

[7] J.M Franco. *Le Data Warehouse, le Data Mining*. Edition Eyrolles, 1996.

[8] R. Kimball. *Entrepôt de données: Guide pratique du concepteur de Data Warehouse*. International Thomson publication, Janvier 1997.

[9] R. Lefébure, G. Venturi. *Le Data Mining*. Edition Eyrolles, 1998.

[10] E. Grislin, D. Donsez. *Data Warehousing/ Data Mining*. Université de Valenciennes (Institut des Sciences et Techniques de Valenciennes), 1999.

[11] G. Gardarin. *Internet/Intranet et bases de données (Data Web, Data Media, Data Warehouse, Data Mining)*. Edition Eyrolles, 2000.

[12] O. Boussaid. *Les entrepôts de données 'Data Warehouse'*. Octobre 2000.

[13] Tuyet-Tram Dang-Ngoc, Georges Gardarin. *Federating heterogeneous data sources with XML*. InProc. Of IASTED IKS Conf 2003: Scottsdale, AZ, USA, November 2003.

[14] Tuyet-Tram Dang-Ngoc, Huaizhong Kou, Georges Gardarin. *Integrating Web information with XML concrete views*. In IASTED DBA (Databases and Applications) 2004, Innsbruck, Austria, February 17-19, 2004.

[15] Sihem Amer-Yahia, Fang Du, Juliana Freire *A comprehensive solution to the XML-to-Relational Mapping problem*. In 6th ACM International Workshop on Web Information and Data Management (WIDM 2004) November 12-13, 2004, Washington, DC, USA.

[16] http://xmlfr.org/actualities/xmlfr/xmlBD.html : traduction de l'article XML and Databases de Patrick Peccatte (Soft Experience)

[17] Sergio Lujan-Mora and Juan Trujillo. *A comprehensive Method for Data Warehouse Design*. In 23rd International Conference On Conceptual Modeling, Shanghai, 2004.

[18] Sergio Lujan-Mora and Juan Trujillo. *Physical Modelling of Data Warehouses using UML*. In Proceedings of the 7th ACM international workshop on Data warehousing and OLAP. Washington, DC, 2004, USA.

[19] Sergio Lujan-Mora and Juan Trujillo. *Multidimensional Modelling with UML Package Diagrams*. In 6th International Conference, Data Warehousing And Knowledge Discovery, Da Wak 2004, Zaragoza.

[20] Manfred A. Jeusfeld, Christoph Quix, Matthias Jarke. *Design and Analysis of Quality Information for Data Warehouses*. In 12th International Conference, Advanced Information Systems Engineering, Caise 2000, Stockholm.

[21] Olivier Teste. *Towards Conceptual Multidimensional Design in Decision Support Systems.* In Proceedings of the 5th East-European Conference on Advances in Databases and Information Systems - ADBIS'01, Vilnius (Lithuania), 25 septembre - 28 septembre 2001.

[22] Mokrane Bouzeghoub, Françoise Fabret, Maja Matulovic-Broqué. *Modeling DataWarehouse Refreshment Process as a Workflow Application.* In 12th International Conference, Advanced Information Systems Engineering, Caise 2000, Stockholm.

[23] Alejandro Gutiérrez, Adriana Marotta. *An Overview of Data Warehouse Design Approaches and Techniques*. In Conf, Université de la République Montevideo, Uruguay. Octobre 2000.

[24] Alain Michard. *XML Langage et applications* EDITION EYROLLES, 1999.

[25] Elliote Rusty Hariold, W. Scott Means. *XML IN A NUTSHELL Manuel de référence.* EDITION O'REILLY, 2001.

[26] Langage de balisage extensible (XML) 1.0. Recommandation du W3C, 10 février 1998. http://www.w3.org/TR/REC-xml.

[27] XML Schema tome0 : Introduction. Recommandation du W3C . xmlfr.org/w3c/TR/xmlschema-0 02 Mai 2001.

[28] http:// xmlfr.org/actualities/xmlfr/040116-0001.

[29] XML Schema Part2 : Datatypes. W3C Recommendation 2 May 2001. http://www.w3.org/TR/2001/REC-xmlschema-2-20010502/.

[30] XML Schema Part1 : Structures. W3C Recommendation 2 May 2001. http://www.w3.org/TR/2001/REC-xmlschema-1-20010502/.

[31] Namespace in XML. World Wide Web Consortium. January 2001.
[32] http:// xmlfr.org/actualities/xmlfr/xmlBD.html.

[33] May Lin, Shun-Sheng Su, Chia-Hung Su. *META DATA EXCHANGE TOOL IN DATA WAREHOUSING*. In IACIS Pacific 2005 Conference Proceedings.

[34] Boris Vrdoljak, Marko Banek and Stefano Rizzi. *Designing Web Warehouses from XML Schemas*. In Proceedings 5th International Conference on Data Warehousing and Knowledge Discovery (DaWaK 2003), 2003.

[35] Andreas S. Maniatis. *OLAP Presentation Modeling with UML and XML*. Knowledge and Database Systems Laboratory (KDBSL) Department of Electrical and Computer Engineering National Technical University of Athens (NTUA) 15780 Athens, Greece. 2003.

[36] Mikael R. Jensen, Thomas H. Moller, Torben Bach Pedersen. *Specifying OLAP Cubes On XML Data.* Technical Report 01-5003, Department of Computer Science, Aalborg University, Created on June 13th, 2001.

[37] Gunnar Auth, Eitel von Maur. *A Software Architecture for XML-based Metadata Interchange in Data Warehouse Systems.* in Proc. Intl. Workshop on XML Data Management (XMLDM), in conjunction with EDBT, Prague, March 2002.

[38] Goksel Aslan, Dennis McLeod. *Semantic heterogeneity resolution in federated databases by metadata implantation and stepwise evolution* . In Computer Science Department, University of Southern California, Los Angeles, CA 90089-0782, USA. Edited by R. King _ Received June 19, 1998 / Accepted April 20, 1999.

[39] Athanasios Vavouras, Stella Gatziu, Klaus R. Dittrich. *Modeling and Executing the Data Warehouse Refreshment Process.* Technical Report 2000.01, January 2000. Department of Information Technology, University of Zurich, Germany.

[40] Panos Vassiliadis, Christoph Quix, Yannis Vassiliou, Matthias Jarke. *DATA WAREHOUSE PROCESS MANAGEMENT.* In Journal Inf. Syst, 2001.

[41] Thilo Maier. *A Formal Model of the ETL Process for OLAP-BasedWeb Usage Analysis.* In Proceedings of the sixth WEBKDD workshop: Webmining and Web Usage Analysis (WEBKDD'04), in conjunction with the 10th ACM SIGKDD conference (KDD'04), Seattle, Washington, USA, August 22, 2004.

[42] Panos Vassiliadis, Alkis Simitsis, Spiros Skiadopoulos. *Conceptual Modeling for ETL Processes.* In Proceedings of DOLAP'02, McLean, Virginia, USA, 2002.

[43] Alkis Simitsis. *Modeling and managing ETL processes*. In DOLAP'02. Washington DC (USA). November 2002.

[44] S. Luján-Mora, P. Vassiliadis, and J. Trujillo. *Data Mapping Diagrams for Data Warehouse Design with UML*. In Proceedings of the 23rd International Conference on Conceptual Modeling, Lecture Notes in Computer Science, Shanghai, China, November 2004.

[45] Mokrane Bouzeghoub, Françoise Fabret, Maja Matulovic-Broqué. *Modeling Data Warehouse Refreshment Process as a Workflow Application*. In Proc. Intl. Workshop on Design and Management of Data Warehouses (DMDW'99), Heidelberg, Germany, 1999.

[46] Christoph Quix. *Repository Support for Data Warehouse Evolution*. In Proceedings of the International Workshop on Design and Management of Data Warehouses (DMDW '99), Heidelberg, Germany, 1999.

[47] Alkis Simitsis, Panos Vassiliadis. *A Methodology for the Conceptual Modeling of ETL Processes.* In Proc. of DSE'03, Velden, Austria, June 17, 2003.

[48] Georges Gardarin, Fei Sha, Tram Dang Ngoc. *XML-based Components for Federating Multiple Heterogeneous Data Sources.* In Proceedings of the 18th International Conference on Conceptual Modeling, 1999.

[49] Kjetil Norvâg. *Temporal XML DataWarehouses: Challenges and Solutions.* In Proceedings of Workshop on Knowledge Foraging for Dynamic Networking of Communities and Economies, Shiraz, Iran, October 2002.

[50] Amar ZERDAZI. *Représentation de schémas de bases de données hétérogènes sous formes de métaschémas XML*. Mémoire de DEA. Université de Marne- la-vallée. Laboratoire d'informatique et de communication (LINC). 2002/2003.

[51] Amélie Marian, Serge Abiteboul, Grégory Cobéna, Laurent Mignet. *Change-Centric Management of Versions in an XML Warehouse*. In Very Large DataBase Conference, Rome - Italy, Septembre 2001.

[52] Saima Iqbal, Julian J. Bunn, Harvey B. Newman. *Distributed Heterogeneous Relational Data Warehouse In A Grid Environment*. In Computing in High Energy and Nuclear Physics, 24-28 March 2003, La Jolla, California.

[53] Verónika Peralta, Adriana Marotta, Raúl Ruggia. *Towards the Automation of Data Warehouse Design*. In CAiSE'03 The 15th Conference on Advanced Information Systems Engineering Velden, Austria, 16 - 20 June, 2003.