N° d'ordre :

République Algérienne Démocratique et Populaire
Ministère de l'Enseignement Supérieur et de la Recherche Scientifique
Université des Sciences et de la Technologie *Houari Boumédiene*
USTHB – ALGER

Faculté d'Electronique et d'Informatique
Département d'Informatique

MEMOIRE

Présenté pour l'obtention du diplôme de

Magister en Informatique

Spécialité : Intelligence Artificielle et Bases de Données Avancées

Par

M^elle EL-GHERS Sabrina

Sujet

Apport de XML dans Un Environnement de Bases de Données Fédérées

Soutenu le 15-12-2004, devant le jury composé de :

Mr	M.N BADACHE	Professeur	USTHB	Président
Mme	Z. ALIMAZIGHI	Maître de Conférence	USTHB	Directeur de thèse
Mme	A. AISSANI	Professeur	USTHB	Examinateur
Mr	A. BELKHEIR	Chargée de Cours	USTHB	Examinateur

Résumé

Les bases de données fédérées sont la nouvelle approche d'intégration des données dans les systèmes d'informations d'aujourd'hui. Une base de données fédérée peut être considérée comme une immense base de données, qui donne à l'utilisateur l'illusion d'une base centralisée et homogène; Or en réalité elle est construite sur une couche de sources de données hétérogènes et disparates sur le réseau.

Théoriquement c'est la solution idéale pour les entreprises, puisque les bases de données fédérées fournissent à travers une interface unique un accès à des données diversifiées tant qu'en format, qu'en emplacement. Cependant la réalisation d'un tel système est loin d'être simple , puisqu'il faut au préalable résoudre tout les problèmes de redondances, d'hétérogénéité et de distribution de ces données.

XML , le langage de balisage du web, représente un excellent langage de modélisation des données, il est suffisamment flexible et expressif pour permettre de capturer l'ensemble de l'information issue d'un tel système.

Notre travail consiste à montrer l'apport de XML dans un environnement de bases de données fédérées. On présente d'abord les propriétés de XML du point de vue des bases de données. On décrit ensuite l'utilisation du modèle XML dans une architecture de base de données fédérée ; enfin on présente une architecture de fédération de données basée sur le modèle XML, et ayant la particularité de donner l'illusion « Tout-relationnel ».

Mot Clés : bases de données fédérées, bases de données distribuées, EAI, services Web, entrepôt de données ,modèle commun, XML,architecture de vues, XQuery, moteur relationnel.

Dédicaces

A ….

Ma très tendre et chère mère pour son soutien irremplaçable, sa patience sans limite , et ses encouragements ;

Mon chère frère Mahdi ;

Mon adorable sœur Nessrine (Techtine), pour sa joie de vivre, qui me remplit de bonheur ;

A mes amies de toujours leila, lamia , Rachid, Nariemene, Widad, Chahra ;

A tout mes ami(e)s ; A Messouda et Feriel .

….Je dédie ce modeste travail.

Sabrina

Remerciements

Je tiens à remercier sincèrement :

Madame Z.Alimazighi, maître de conférences à l'USTHB, pour m'avoir proposé un sujet passionnant, pour sa disponibilité et pour les précieux conseils qu'elle m'a prodigué au cours de ce travail et pour ses encouragements durant mes études de graduation et post-graduation.

Monsieur N. Badache, Professeur à l'USTHB , pour avoir accepté de présider le jury.

Madame A. Aissani-Mokhtari, Professeur à l'USTHB, pour ses encouragements durant mes études de post graduation et pour avoir accepté d'être membre du jury.

Monsieur A. Belkheir, Chargé de cours à l'USTHB, pour ses encouragements et surtout son indulgence durant mes étude graduation et post graduation.

J'adresse mes plus chaleureux remerciements à tous mes enseignants de la poste graduation ainsi que ceux de la graduation de l'institut informatique de l'USTHB.

Je remercie également Leila pour l'aide qu'elle m'a apporté pour réaliser ce document et le soutien moral sans faille qu'elle m'a toujours exprimé.

Enfin, j'adresse mes plus vifs remerciements à toutes les personnes qui m'ont soutenu et qui m'ont encouragé.

TABLE DES MATIERES

TABLE DES ILLUSTRATIONS

CHAPITRE II : LES BASES DE DONNEES DISTRIBUEES.

CHAPITRE III: APPROCHES D'INTEGRATIONS

CHAPITRE IV : XML COMME MODELE DE DONNEES

CHAPITRE V : LES SYSTEMES DE MEDIATION BASES XML

CHAPITRE VI : UNE ARCHITECTURE DE MEDIATION :TOUT-RELATIONNEL

TABLEAUX

Chapitre 1 : Introduction

1. Domaine de la thèse

Aujourd'hui , l'informatique au sein des organisations est marquée par l'apparition de nouvelles technologies visant non seulement à renforcer les solutions techniques existantes mais aussi de leur rajouter une couche métier ; Le but étant de répondre aux besoins utilisateurs, devenus de plus en plus importants et ceci de manière efficace et surtout en temps réel.

Disposant de ces outils divers, les entreprises tentent de mettre en place des systèmes d'information transversaux à la fois entre les services et inter-entreprises; le but étant de pouvoir interconnecter un ensemble d'applications et de services afin de répondre aux attentes des utilisateurs et aux nouveaux modes d'organisation.

Afin de permettre une intégration des données, une refonte des systèmes d'information s'avère alors nécessaire . Les solutions escomptées devront non seulement prendre en compte l'existant, mais aussi assurer l'accès en temps réel à plusieurs sources de données hétérogènes et réparties sur l'ensemble du réseau, tout en tirant profit des technologies web (publication, portail, et ... source HTML).

Par le passé **[GAR 91]**, les systèmes de gestion de bases de données distribuées ont tenté de répondre à ces contraintes : utiliser une interface commune pour accéder à des données hétérogènes et distribuées. Des prouesses dans le domaine des requêtes distribuées ont été réalisées, mais le coût de déploiement ainsi que les faibles performances de ces requêtes ont fait que ces systèmes ont vite été abandonnés à peine expérimentés.

Enfin, grâce à des progrès récents dans tous les domaines (SGBDR, réseau, etc.),on s'intéresse de nouveau à cette approche ; cette fois ci, elle est appelée « *bases de données fédérées* » ou même « *base de données virtuelle* » (et même aussi EII pour Enterprise Informations Integration). Il n'est plus question de réinventer les bases de données fédérées mais plutôt de permettre l'accès « temps réel » à des sources de données diversifiées tant en emplacement qu'en format.

Un système de bases de données fédérées est un système qui permet d'*intégrer plusieurs sources de données, hétérogènes* et *distribuées,* implémentées avec des gestionnaires différents, souvent avec des modèles différents (Fichiers, réseaux, relationnels, Objet, …etc.). Cependant, la réalisation d'un tel système nécessite l'utilisation d'un modèle de données commun, capable d'exprimer les différents modèles de la fédération, et donner ainsi une vue globale uniforme de toutes les données appartenant au système.

La popularité de XML comme format de transfert de données sur le web lui permet d'être le candidat idéal pour être le modèle commun dans un système de fédération de données. En effet, il présente l'avantage d'être suffisamment flexible pour exprimer n'importe quelle source de données, à savoir les bases de données relationnelles, les fichiers plats, des pages HTML…etc.

C'est dans ce contexte que se situe le domaine d'étude de cette thèse. Plus particulièrement dans l'apport de XML dans un environnement de fédération de données.

2. Problématique

Le développement d'un système de base de données fédérée repose essentiellement sur le concept d'intégration de données issues de sources hétérogènes et distribuées. En d'autre termes, pour pouvoir donner à l'utilisateur, l'illusion d'un système central homogène, au dessus de plusieurs sources différentes, il est essentiel d'une part ,d'unifier les modèles de données des sources sous jacentes en un modèle commun ; et d'autre part, d'unifier les langages de requêtes en un langage commun. Aujourd'hui, pour répondre à ces besoins deux approches s'affrontent :

L'approche *« WareHouse »*, qui consiste dans la matérialisation des données dans une base intégrée ***[GAR 03a]*** appelée *entrepôts de données*. L'intégration se fait de manière périodique grâce aux techniques telles que les ETL (*Extraction,Transformation,Loading*).

La seconde approche dite celle des *données fédérées* où les données, contrairement à l'approche précédente, ne sont pas matérialisées dans un SGBD. L'intégration se fait sur demande pour répondre aux requêtes, en temps réel.

Le travail réalisé dans cette thèse consiste à montrer l'apport du langage XML dans l'approche dite des données fédérées ou système à médiateur.

En effet, la fédération de données hétérogènes provenant de sources multiples nécessite un modèle d'échange commun. Parmi les modèles de conception de bases de données distribuées, le modèle relationnel fut le plus répondu dans les années 80 **[GAR 91]**, des prototypes ont été réalisés comme R*, DB2 distribué ou Oracle * …etc. Durant les années 90, une nouvelle technologie est apparue, celle des médiateurs (*Middelaware*) comme OLE DB de Microsoft…etc. Cependant, ces techniques ont vite montré leurs limites surtout quand il s'agit du support des documents et du Web.

Aujourd'hui XML semble être le modèle le plus approprié. En effet, il présente les caractéristiques suivantes :

- Un modèle complet : les schéma XML peuvent être utilisés comme des schémas de conception complets, permettant d'exprimer les relations, les attributs, l'héritage, le typage, l'hiérarchisation…etc.
- Un schéma d'échange explicite inclus dans les données : possibilité d'unification des noms en utilisant la notion *des espaces de noms.*
- Une simplicité de génération depuis des formats divers : SQL, fichiers légataires, documents (HTML,…etc.)
- XML à un support pour les requêtes : XQuery.

Le but de notre thèse est de montrer les différentes solutions qu'apporte le langage XML comme modèle de données commun dans un système de base de données fédérée. Une architecture de médiation inspirée des architectures existantes ***[SHA 01] [MAN 00] [MAN 01]***et basée sur le modèle XML est proposée. Elle présente la particularité de donner l'illusion d'un système « Tout –Relationnel » en acceptant , à la différence des autres systèmes , des requêtes utilisateurs formulées dans le langage SQL.

3. Plan de la thèse

La présente thèse est structurée comme suit :

Le *Chapitre 2* présente un état de l'art des systèmes de gestions des bases de données distribuées, tels qu'ils ont été élaborés au début des années 80, nous parlerons des objectifs de ces systèmes, de leur classification, des architectures les plus couramment utilisées, des techniques de fragmentation utilisées. Ainsi que leurs fonctionnalités les plus importantes comme le contrôle sémantique, la gestion des transactions distribuées.

Dans le *Chapitre 3* , on aborde les nouvelles approches de fédération de données. A savoir les outils EAI *(Eentreprise Application Integration),* les services webs et les DataWarehouses. Ces trois approches représentent la seconde génération de solution à s'attaquer au problème d'intégration de données issues de sources hétérogènes et distribuées. Chacune de ces approches est détaillée, entre autre on décrira l'architecture ,les fonctionnalités et les protocoles de communication. Finalement, une introduction au concept EII (*Entreprise Information Integration*) à travers une comparaison avec les autres approches conclut le chapitre.

Le *Chapitre 4* présente un état de l'art sur les données semi structurées, entre autre XML. Il précise les éléments de base de ce langage, qui sont relatifs au domaine des bases de données. D'abord les concepts des données semi structurées sont présentés tels que la grammaire , les objectifs, et surtout leur modélisation. Ensuite c'est le langage XML qui est détaillé à travers ses éléments de syntaxes, le modèle de données qu'il infère , tels les schémas XML-Schéma et DTDs.

Dans *le chapitre 5* , on aborde la médiation basée XML proprement dite, à travers le langage de requête XQuery, qui est le langage de requête standard pour les données au format XML. tout d'abord on présente un état de l'art sur les langages de requêtes des données XML entre autre OEM-QL, Xpath, XML-QL et XQL. La deuxième partie de ce chapitre comporte une introduction aux bases de données fédérées ainsi qu'aux différentes architectures existantes.

Le C*hapitre 6* présente l'architecture que nous préconisons , les différents composants ainsi que leurs fonctions sont détaillés. Les plus importantes techniques de mises en œuvre sont aussi décrites, plus particulièrement l'intégration des schémas, la traduction des requêtes XQuery vers XML ,et enfin l'exportation de données relationnelles en XML.

Chapitre 2 : les systèmes de bases de données distribuées

1. Introduction

Devant la prolifération des réseaux, les concepteurs de systèmes de gestions de bases de données ont vu la nécessité de développer des systèmes de plus en plus performants, capables de fournir aux utilisateurs des données correctes à partir de plusieurs sources hétérogènes et ceci de manière transparente en utilisant une requête unique et simple.

En effet, les développements remarquables dans les domaines des réseaux de communication et des bases de données, ont permis à l'approche *base de données distribuée (répartie)* de devenir une solution alternative à la centralisation ***[GAR 91]***.Une base de données distribuée (*distributed database*) est une collection de données logiquement corrélées, et physiquement réparties sur plusieurs machines interconnectées sur un réseau ***[GAR 91].***

Dans ce chapitre, nous étudions en détail la technologie des bases de données dites *distribuées* ou *réparties*. Dans la section (2), on définit les systèmes de bases de données distribuées ; dans la section (3), on détaille les objectifs les plus pertinents de ces systèmes. La section (4) présente les critères de classification des SGBDs distribués. L'architecture est décrite à travers les sections (5) et (6). Les différentes techniques de distribution sont détaillées dans la section (7). La section (8) aborde les fonctionnalités les plus importantes de tels systèmes. Les bases de données fédérées sont décrites en détails dans la section (9). Enfin, on conclut dans la section (10).

2. Définition d'un système de bases de données Distribuées

2.1 Le Concept de « Distribution »

- *Un système de traitement distribué :* est une *unité d'exécution* de programmes *autonomes,* éventuellement *hétérogènes, reliés par un réseau* de communication et *coopérants* à la réalisation de taches. ***[BOU 03]***

- *Une base de données distribuée :*est une collection de bases de données logiquement reliées et physiquement distribuées sur un réseau.***[BOU 03]***.G.Gardarin la définit dans ***[GAR 91]*** comme suit : C'est un ensemble de bases de données coopérantes, chacune résidente sur un site différent, vues et manipulées par l'utilisateur comme une seule base de données centralisée.

- *Un système de gestion de bases de données distribuées :* est un système logiciel qui fournit une interface logique unique aux différentes bases de données distribuées.

2.2 Motivation de la distribution

Les bases de données distribuées ont été les premières réponses à un vieux rêve qui hante l'industrie informatique depuis des décennies : permettre d'accéder à des données différentes, en utilisant une interface unique, et ceci quelque soit l'emplacement et le format de celles ci. Les systèmes de bases de données distribuées tentent de répondre aux exigences suivantes : ***[BOU 03]***

- *Distribution naturelle :* avec la prolifération des réseaux, les données ne sont plus centralisées mais plutôt distribuées sur celui ci.

- *Fiabilité :* en ayant des données dupliquées sur plusieurs sites, on permet leur disponibilité en cas de pannes.

- *Economie :* les systèmes distribués permettent un gain en ressources réseau, vu le traitement local des données. Le traitement parallèle des requêtes permet également un gain en en performance.

- *Partage des données et des ressources :* la base de données distribuée est ainsi partagée et enrichie à fur et à mesure que le réseau s'étend.

- *Autonomie :* la distribution permet aux différentes structures d'une même entreprise de conserver leur spécificité.

2.3 Inconvénients de la distribution

Parmi les inconvénients majeurs des systèmes distribués, on peut citer ***[GAR 91][BOU 03]***:

- Le manque d'expérience des équipes de développement et des utilisateurs finaux.
- Coûts de déploiement (de communication, d'adaptation des applications et des hommes).
- Conception et développement complexe pour des résultats et des performances insignifiantes.

3. Objectifs des SGBDs Distribués

Il n'existe pas de système de gestion de base de données distribuée capable d'atteindre la totalité des objectifs de conception de ces systèmes. En effet, en fonction du marché d'applications visé, un SGBD réparti n'atteindra qu'un sous ensemble de ces objectifs ***[GAR 91],*** parmi lesquels on peut citer :

3. 1 La transparence

Dans un environnement distribué, il est plus simple pour un utilisateur d'avoir une vue unique et intuitive sur le système, comme s'il s'agissait d'un système local. Ceci implique la transparence de tous les traitements distants à l'utilisateur, à savoir : ***[GAR 91]***

- *Transparence de consultation :* Une opération de consultation doit donner le même résultat quelque soit le site.

- *Transparence de mise à jour :* Emise sur n'importe quel site, une mise à jour impliquant plusieurs sites, donnera toujours le même résultat.

- *Transparence de schéma :*Les schémas de tous les sites peuvent être visibles sur n'importe quel site.

- *Transparence de performance :*Les performances d'exécution des requêtes doivent être les mêmes quelque soit le site d'émission.

- *Transparence de transaction :*Lors de l'exécution des transactions réparties concurrentes, le cohérence de la base de données distribuée doit être maintenue.

- *Transparence de copies :*Les copies des données doivent être faites sans l'intervention de l'utilisateur final.

En pratique, assurer la transparence dans un système distribué, revient à traiter les problèmes suivants :

3.1.1 Indépendance à la localisation

L'*indépendance à la localisation de données* cache le fait que les données ne résident pas nécessairement sur le site utilisateur. Celui ci peut ignorer la localisation des données. L'information concernant la localisation des données, est alors maintenue dans le *dictionnaire de données,* et consultée par le SGBD distribué pour déterminer la localisation des relations impliquées dans la requête des utilisateurs. ***[GAR 91]***

3.1.2 Nomination des objets (*Namming*)

Dans un système distribué, les noms des entités manipulées peuvent poser un sérieux problème : ils doivent être unique à travers tout le système ***[BOU 03] [BUK 96]***. Pour pallier à ce problème plusieurs techniques existent : ***[BUK 96]***

- *Un serveur de noms centralisé :* Il génère pour chaque entité du système , un nom et un identifiant unique et global à travers tout le système. Cet approche présente l'inconvénient de surcharge du serveur local , ainsi que le blocage de tout le système en cas de panne de celui ci ; de plus elle nuit à l'autonomie du système distribué.

- *Utilisation d'alias :* Une autre solution serait de fournir un ensemble de règles de passage (*un système de mapping)* entre les noms locaux et globaux logiques. Cependant, ceci est difficile à maintenir dans un environnement global ou des entités individuelles sont crées, modifiées et supprimées constamment.

3.1.3 Traduction Données/Fonctions

Les bases de données distribuées doivent assurer l'indépendance aux SGBDs locaux, c'est à dire cacher le fait, que les SGBDs locaux peuvent être différents (modèle de données et langages associés sont différents). ***[GAR 91]***

Par conséquent, quand un système utilise des données à partir des sources multiples et distribuées, une traduction du modèle de données et du langage de chaque SGBD local dans ceux de chaque autre SGBD local doit être effectuée. La solution la plus courante consiste à

employer un modèle pivot (et son langage associé), et un traducteur entre ce modèle pivot et le modèle de chaque SGBD local. ***[GAR 91].*** Pour respecter le principe de transparence, il est évident que ces transformations doivent êtres exécutées de manière automatique. ***[BUK 96]***

3.1.4 Navigation globale

Les utilisateurs ont besoin de naviguer dans l'espace des Données /Fonctions global. Le SGBD distribué doit fournir une interface capable de localiser l'information recherchée. Exemple : Un répertoire de style hypertexte peut être utilisé afin de représenter les fonctionnalités globales du système.***[BUK 96]***

3.1.5 Fiabilité distribuée

Dans le cas d'un système unique, l'apparition d'une panne peut bloquer la totalité des activités du système ; par conséquence, on ne peut y accéder jusqu'au redémarrage de celui ci. Dans un système distribué, on peut envisager de répliquer (dupliquer) les données, de façon à permettre au système global de router la requête utilisateur à une source alternative, quand la source principale ne fonctionne plus. Ceci peut engendrer une diminution dans les performances du système, cependant en cas de panne, il peut fonctionner de manière normale. Quand il s'agit d'une requête qui demande plusieurs sources distantes et que l'une d'elle ne fonctionne plus, le système distribué peut fournir une réponse partielle. Le routage des requêtes et le calcul partiel des réponses doivent être des fonctions automatiques du système. ***[BUK 96]***

3. 2 L'autonomie

L'autonomie est la capacité des systèmes SGBDs locaux de traiter les requêtes locales de manière totalement indépendante du système global. En effet, l'utilisateur peut intégrer ou pas le système distribué, sans que cela affecte les performances du SGBD local. Pour ce faire, chaque site doit disposer localement de toutes les informations qui se trouvent dans le *dictionnaire de données* afin d'être autonome ,sans avoir à compter sur *un dictionnaire de données global centralisé*. ***[GAR 91]***

L'autonomie est étroitement liée à la transparence, elle mesure le degré de dépendance vis à vis du système distribué. Cependant, gagner en transparence fait perdre en autonomie et vice versa.

3.3 La complexité de conception

Les systèmes distribués sont des systèmes complexes et qui requièrent l'implémentation de services complexes d'où la nécessiter d'utiliser : ***[BUK 96]***

- *L'indexation :* Les ressources du système distribué global doivent être indexées à des niveaux d'abstractions variés. Ceci permet à l'utilisateur de voir clairement une image globale de l'information, de pouvoir zoomer dedans selon ce qu'il désire connaître.
- *Multiples modèles accès /présentation :* Le système de base de données distribuée doit prendre en charge les différentes transformations entre les modèles de données et les

langages d'accès. Il doit être capable d'intégrer les représentations hétérogènes et de présenter une interface unique, uniforme aux utilisateurs globaux.

3.4 Extensibilité (*Scalability*)

C'est la capacité d'augmentation incrémentale, par introduction de nouveaux sites dans le réseau, avec un impact minimal sur les bases de données locales et les programmes d'applications existantes ***[GAR 91]***

Le système de gestion de bases de données distribué doit être capable d'intégrer les nouveaux nœuds ajoutés au réseau. l'interface de l'utilisateur global doit être assez évolutive, pour permettre d'intégrer de nouvelles fonctionnalités sans détruire la gestion de l'interface existante. D'autre part , le nœud local (SGBD local) doit continuer à être autonome, tout en dédiant certaines de ses ressources au SGBD distribué. ***[BUK 96]***

3.5 L'interface utilisateur

L'interface utilisateur dans un système distribué est très difficile à réaliser, car elle doit être intuitive et simple pour les utilisateurs novices, puissante et efficace pour ceux expérimentés.

4. Classification des SGBDs distribuées

Les systèmes de bases de données distribués sont classés selon les critères suivants : ***[BOU 03]***

- *L'autonomie :* Elle mesure le degré d'indépendance des différents SGBDs locaux qui participent à la distribution ; elle est calculée en fonction des échanges inter-sites, ainsi que de la capacité de calculer des transactions indépendamment des autres sites. Elle implique aussi le degré de contrôle du SGBD global sur les SGBDs locaux.

- *La Distribution :* Elle mesure le degré de distribution des données. (distribué ou pas).

- *L'hétérogénéité* **:** elle consiste dans l'hétérogénéité des facteurs suivants :
  - Hardware et protocole réseau.
  - Modèles de données.
  - Langages de données.

Parmi les multitudes de systèmes que peut engendrer cette classification, on peut en citer l'ensemble suivant :

- *Les Systèmes de bases de données distribuées (ou réparties):* est une collection de système de gestion de bases de données locaux, éventuellement hétérogènes et qui coopèrent pour fournir une interface globale et uniforme. ***[GAR 91][BUK 96][BOU 03]***. C'est un ensemble de bases de données fortement couplées, administrées et manipulées comme une seule bases de données. ***[GAR 91]***

- *Les Systèmes Multibases (Systèmes de bases de données fédérées) :* Une MultiBase de données intègre les systèmes de gestions de bases de données (SGBD) locaux, pré existants et hétérogènes ***[BUK 96].*** C'est un ensemble de bases de données faiblement

couplées et autonomes qu'u utilisateur peut manipuler à l'aide d'un langage spécifique. ***[GAR 91]***

- *Les systèmes inter opérables :* Ce type de système fournit des formats et des protocoles pour intégrer des données entre des systèmes locaux mais ne fournit pas des fonctionnalités globales. ***[BUK 96]***

5. Architecture des SGBDs distribués

5.1 Approches de conception

Lors de la conception d'une base de données distribuée, on procède selon les deux approches suivantes **: *[GAR 91]***

5.1.1 Approche Descendante

Cette approche consiste dans la répartition d'une base de données centralisée, en une collection de données appartenant logiquement au même système, mais physiquement étalée sur plusieurs sites. Dans ce cas la base de données distribuée est dite *homogène.* La démarche *descendante* comprend trois étapes (figure 2.1):

- La conception du SGBD global
- La fragmentation du SGBD à travers les différents sites.
- L'allocation des sites au profit des fragments.

Cette démarche facilite l'ajout d'un nouveau site dans le réseau , cependant, une modification du schéma global peut entraîner la reprise de tout le processus de conception.

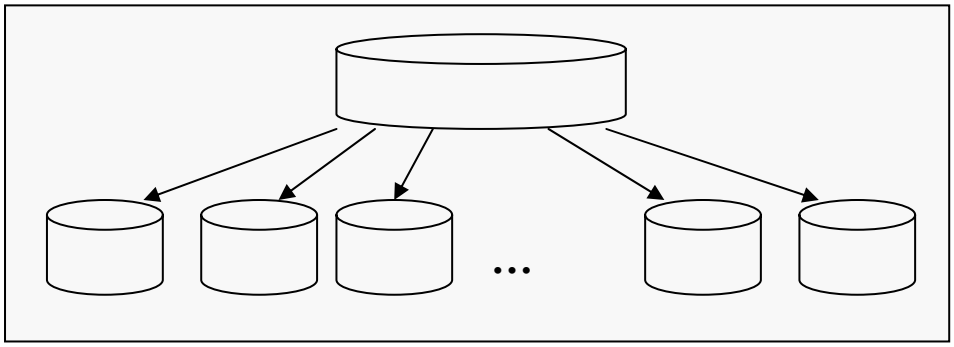

FIG. 2.1 - Approche descendante

Exemple : R* de IBM (1979,1984). Ce SGBD réparti consiste dans plusieurs sites autonomes dotés du même SGBD . ***[GAR 91]***

5.1.2 Approche Ascendante

Dans cette approche, la base de données distribuée est construite au dessus d'une couche de base de données pré existantes (figure 2.2).

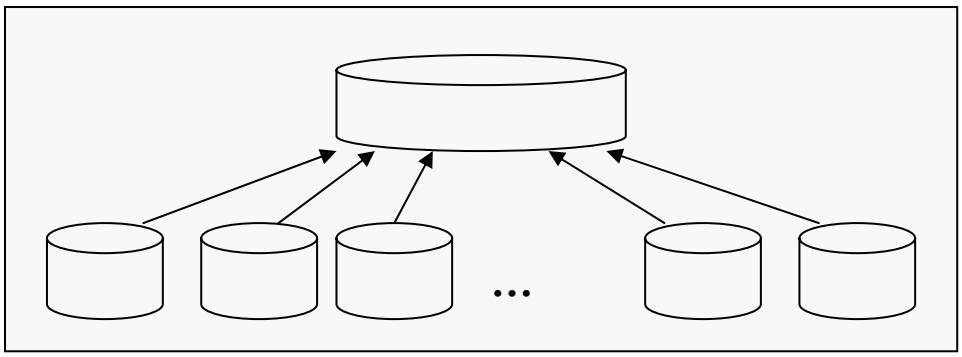

FIG 2.2 - Approche Ascendante

Dans ce contexte, la base de données distribuée est dite *Multibases de données.* Elle peut être *homogène* si les SGBDs locaux sont les mêmes. Elle est dite *hétérogène* dans le cas contraire. La caractéristique essentielle d'une base de données distribuée hétérogène est d'intégrer des bases de données locales, existantes et indépendantes, gérées par des SGBDs différents**.** ***[GAR 91]*** Dans notre étude , nous nous intéressons particulièrement à ce dernier type appelé aussi *Base de données fédérée*.

5.2 Architecture des schémas

Une base de données distribuée est organisée selon une architecture dite *des schémas.* L'architecture des schémas présentée dans la figure 2.3 est décrite en termes de niveaux de schémas (les boites) et de transformation de niveau de schémas (les lignes reliant les boites). Ainsi les niveaux de schémas qui décrivent la base de données distribuée indépendamment de toute base locale sont appelés *globaux.* Une requête exprimée sur le schéma global est dite *requête distribuée (répartie).* On distingue trois niveaux de schémas globaux ***[GAR 91]* :**

- *Schémas externes globaux :* chacun des schémas externes globaux supporte une vue particulière des données. Ces schémas assurent l'indépendance logique.
- *Schéma conceptuel global :* il définit toutes les relations appartenant à la base de données distribuée.
- *Schéma de placement :* ce schéma précise la façon dont les relations sont placées sur les différents sites dans le réseau (information sur la localisation, fragmentation et duplication des données).

Les *schémas locaux* quant à eux, ils décrivent une base de données locale. Il est à noter qu'il existe autant de schémas locaux que de sites dans le réseau. Une requête exprimée sur un schéma local est dite *requête locale.* Dans l'architecture des schémas , il existe trois niveaux de schémas locaux :

- *Le schéma local interne, le schéma local conceptuel :* sont identiques à ceux d'une base de données centralisée.
- *Le schéma local externe :* il représente les fragments tels qu'ils sont décrits dans le schéma de placement, comme des vues de la base locale.

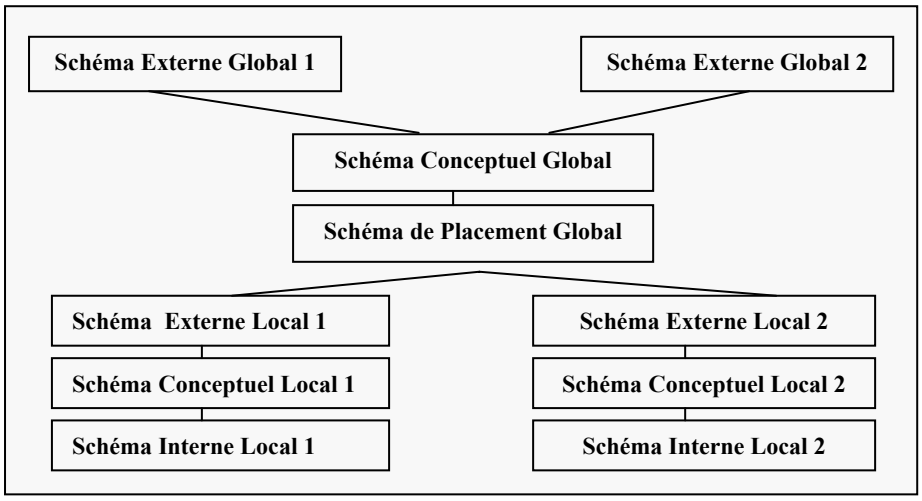

FIG. 2.3 - Architecture des schémas d'une base de données distribuée

6. Architecture fonctionnelle d'un SGBD distribué

Les fonctions d'un système de gestion de base de données distribuée sont groupées dans deux composants systèmes (figure 2.4) ***[GAR 91][BOU 03]*** **:**

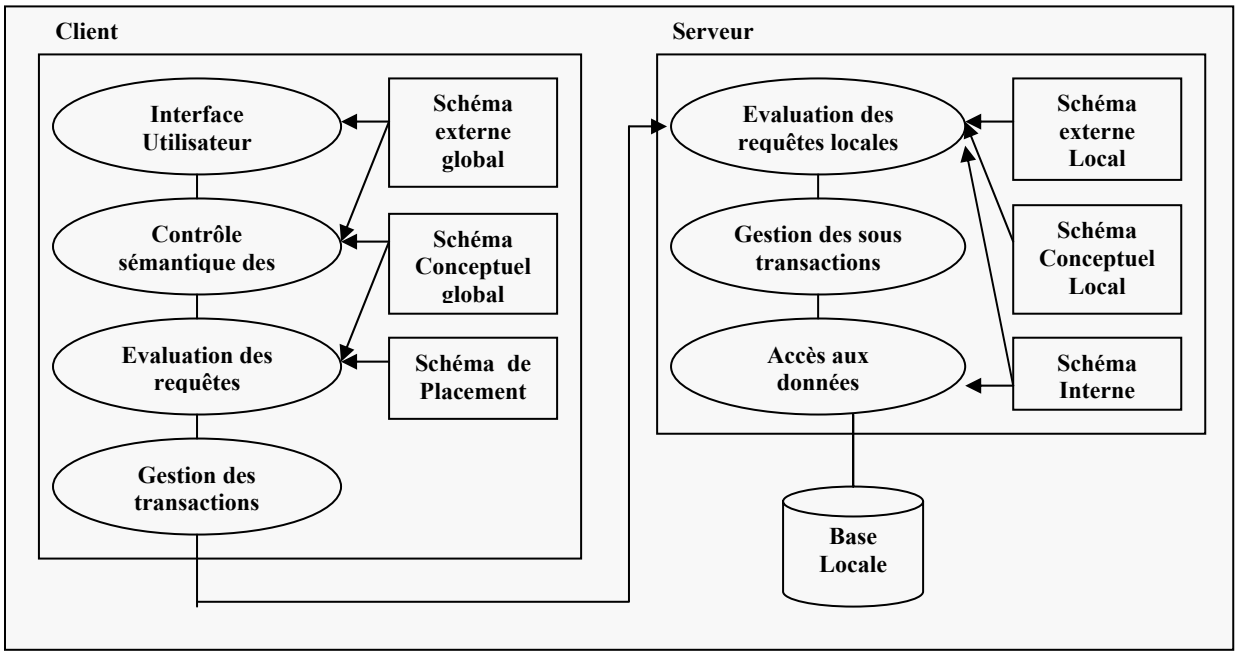

FIG. 2. 4 - Architecture Fonctionnelle d'un SGBD Distribué

- *Le processeur d'applications (Client ou gérant d'applications) :* présent sur chaque site, il intègre les quatre fonctionnalités suivantes :

    1. *L'interface utilisateur :* reçoit la requête utilisateur ,vérifie que les données spécifiées sont conformes au schéma externe, et après exécution de la requête, restitue les résultats à l'utilisateur.

2. *Le contrôle sémantique des données :* ce composant transforme la requête exprimée sur des vues, du schéma externe global, en une requête exprimée sur des relations définies dans le schéma conceptuel global, puis ajoute les contrôles d'autorisations et d'intégrité sémantiques.

3. *L'évaluation des requêtes :* ce composant transforme la requête répartie issue de l'étape précédente en un plan *d'exécution réparti.* Ce plan d'exécution est une séquence optimisée des requêtes locales, synchronisées par les échanges inter-sites.

4. *La gestion de transaction :* à ce niveau ,on s'occupe de la coordination et de l'exécution des transactions , entre autre de :
   a) Traduction *(Translation)* des transactions.
   b) Contrôle des concurrence distribuées.
   c) Contrôle de l'exécution de la transaction distribué ainsi que sa validité.

- *Le processeur de données (Serveur ou gérant de données) :* ce composant s'occupe de trois fonctionnalités :

1. *L'évaluation des requêtes locales :* la requête exprimée dans le schéma de placement est traduite en une requête dans le schéma externe local. Dans le cas d'une base de donnée hétérogènes, ceci correspond à une traduction du modèle\langage *pivot* au modèle\langage local. Les suites de transformations sont classiques : vers une requête optimisée exprimée dans le schéma conceptuel local et finalement vers le schéma interne local.

2. *La gestion des sous transactions :* ce composant coopère avec le *processeur d'applications* qui initie la transaction et les autres *processeurs de données* afin de synchroniser l'exécution des sous transactions issues de la même transaction globale.

3. *L'accès aux données :*Mise à jour de la base de données locale (fonctions normales de gestion de données d'un SGBD).

7. Les techniques de distribution

7.1 La Duplication

Dans une base de données distribuée, la duplication des données représente une solution au problème de fiabilité ***[GAR 91].*** Elle consiste dans la création de plusieurs copies des mêmes données, on peut faire une *Duplication totale* ou *Partielle* des données.

La duplication présente les avantages suivants : ***[GAR 91].***

- *Disponibilité des données* : Une Relation R peut être atteinte à partir de n'importe quel site. Si un site devient non opérationnel, à la suite d'une panne, la copie des données est toujours accessible sur un autre site.
- *Parallélisme des Traitements en lecture:* Plusieurs sites peuvent travailler en parallèle sur une relation R. Ce qui augmente les performances d'accès.

- *Diminution du Flux sur le réseau :* quand on copie des données sur un site, où elles sont le plus fréquemment utilisées, on favorise l'accès local en évitant l'accès réseau.

Par contre , la duplication présente les inconvénients suivants :

- *Coût des mises à jour élevé :* Toute mise à jour doit être appliquée à l'ensemble des sites concernés par la relation, pour garantir la consistance des répliques des relations.
- *Augmentation du volume de la base de données.*

7.2 La Fragmentation

La fragmentation sert à diviser une relation globale en unités logiques d'allocation (les fragments), qui peuvent être placées de façon optimale dans la base de données distribuée. Afin de préserver la cohérence sémantique de la base, la fragmentation doit respecter les règles suivantes : ***[GAR 91] [BOU 02a] [BOU 02b].***

- *Décomposition sans perte :* chaque élément de données appartenant à la relation globale appartient aussi à un ou plusieurs de ses fragments.
- *Reconstruction :* d'une relation globale à partir de ses fragments doit être toujours possible.
- *Non duplication :* impose que les fragments sont disjoints dans le cas de la fragmentation horizontale.

La fragmentation d'une relation R est le découpage de celle ci en *Fragments R1,R2,…Rn.* Dont chaque fragment contient des données de R.

7.2.1 Fragmentation Horizontale

Supposons qu'on a une relation R, subdivisée en un ensemble de sous relations r1,r2,…rn de même schéma que R. Une *fragmentation horizontale* (figure 2.5) sur R, se fait par une restriction sur R tel que : *ri = Restrict (R/Pi)* ou *Pi* est la condition de la restriction. La reconstitution de R se fait par l'opérateur Union tels que *R = Union(ri) ; (i=1 à n).*

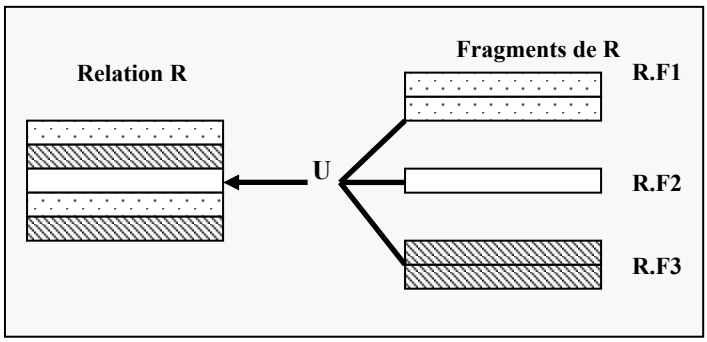

FIG. 2.5 - Fragmentation Horizontale

7.2.2 Fragmentation Verticale

La fragmentation verticale de le relation R (Figure 2.6), consiste dans la décomposition de la relation R en plusieurs sous ensembles r1,r2,...rn tel que chaque fragment *ri = Project (R/ri)* . La reconstruction de R se fait par l'opérateur Jointure : *R= r1 JOIN r2 JOIN r3...JOIN rn.)****[BOU 02a] [BOU 02b]***

Il est nécessaire de rajouter un attribut à la relation R, appelé *Identifiant (Tuple -Id).* Il correspond à l'adresse logique ou physique d'un tuple. Si lors de la conception, on utilise un identifiant *interne,* on peut assurer une indépendance entre les données et les programmes, telles que les modifications des emplacements des tuples seraient sans incidence sur les programmes.***[BOU 03].***

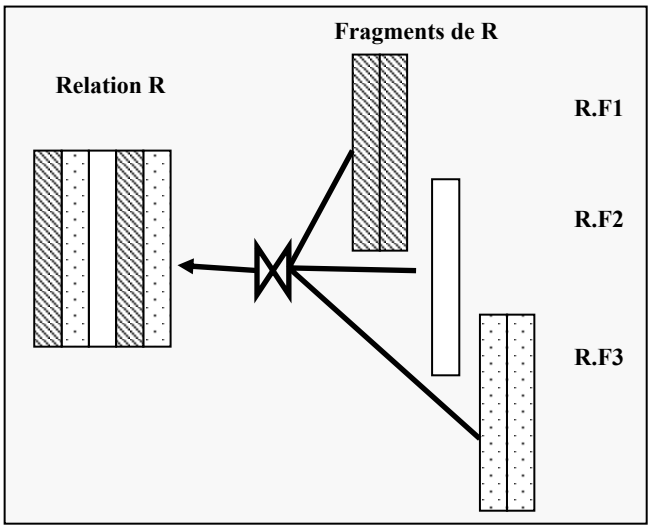

FIG. 2.6 - Fragmentation Verticale de R

7.2.3 Fragmentation mixte et duplication

Dans ce type de fragmentation, chaque fragment est obtenu par applications successives des fragmentations horizontales et verticales. On peut aussi avoir des fragments de duplicata ou duplications de fragments***.[BOU 02a]***

7.3 La localisation des données

Dans un système de base de données distribuée, où les données sont éparpillées sur le réseau sous forme de fragments ou de copies, on convient d'utiliser un système de *Nommage* d'entités tel que ***[BUK 96]*** **:**

- Une entité contenue dans un fragment i est désignée par : *Désignation*_Entité**.** *Fi.*
- Une entité contenue dans une copie i est désignée par : *Désignation_Entité***.** *Ci.*

Exemple : Produit.F3.C4

Pour permettre la localisation des entités à travers tout le système , on utilise le principe d'*Alias* sans nom du site. Ceci permet de changer la localisation des données sans incidence sur les programmes et les utilisateurs. ***[BUK 96]***

8. Fonctionnalités d'un SGBD distribué

Les fonctions spécifiques d'un SGBD distribué sont ***[GAR 91] [BUK 96]*** **:**

- La gestion d'un *dictionnaire de données* global qui maintient l'information relative aux données distribuées.
- Le contrôle sémantique des données distribuées.
- L'évaluation des requêtes distribuées.
- La gestion des transactions distribuées.

8.1 Le dictionnaire de données

Le dictionnaire de données d'une base de données distribuée, est une base de données contenant les *méta-données* qui décrivent les données de la base. Il contient, entre autre , les descriptions des données globales, des informations sur le placement des données et le contrôle sémantique des données. ***[BUK 96]***

Le dictionnaire de données est géré comme une base de données distribuée. Pour ce faire, trois approches d'organisation sont utilisées : ***[BUK 96]***

- *Centralisation :* Le dictionnaire existe sur un site unique, ce qui simplifie l'administration de la base de données distribuée, ainsi que les mises à jour des schémas ; par contre, les traitements de requêtes initiées sur d'autres sites, peuvent s'avérer coûteux, puisque celles ci requièrent un accès réparti au site central.

- *Duplication :*gère une copie du dictionnaire sur chaque site, évitant ainsi les inconvénients de l'approche précédente, cependant les mises à jour du dictionnaire doivent être répliquées sur tous les sites.

- *Répartition :* fragmente le dictionnaire en dictionnaires locaux, chacun contenant uniquement les informations liées aux données du site où il est stocké.

8.2 Le contrôle sémantique des données distribuées

Le contrôle sémantique des données consiste dans la gestion des vues, le contrôle des autorisations et le contrôle d'intégrité sémantique des données. Ainsi, un système de SGBD distribué assurant ces fonctionnalités, garantit que les utilisateurs autorisés exécutent des opérations correctes qui préservent la cohérence de la base de données***[BUK 96]*** .

8.2.1 Les vues

Dans un SGBD distribué, on peut utiliser des vues définies sur des fragments répartis. Quand une requête distribuée est exécutée sur une vue, elle peut être traitée des trois manières suivantes : ***[BOU 02a] [GAR 91]***

- *Par réécriture* **:** La requête sur le Vue V est transformée en une requête sur une relation répartie R, c'est à dire sur les fragments correspondants.

- *Par la méthode des vues concrètes :*On crée des vues réellement , elles sont calculées, stockées et mises à jour de manière systématique. Elles peuvent être dupliquées sur le site où leur utilisation est fréquente. Les vues concrètes évitent le calcul d'une vue à chaque requête, qui peut s'avérer coûteux par la suite.

- *Par la méthode des clichés :* Un cliché étant une vue concrète qui n'est mise à jour que périodiquement, ceci résout le problème de mises à jour coûteuses des vues concrètes. On utilise une *Delta relation ,*qui contient les tuples du cliché à modifier ; ce qui réduit le coût de mise à jour.

8.2.2 Les Autorisations

Le contrôle des autorisations consiste dans l'identification et l'authentification des utilisateurs non locaux. Pour ce faire, il existe deux approches possibles : ***[GAR 91]***

- *Utilisation du dictionnaire de données :* dans cette approche, chaque sous transaction indique le nom utilisateur et le mot de passe, afin d'être acceptée comme étant locale. Ces informations sont stockées dans le dictionnaire de données global, qui doit nécessairement être dupliqué sur chaque site.

- *Authentification des sites :* chaque site de la base de données distribuée, doit être identifié et authentifié. Dans ce cas les sous transactions d'une transaction initiée à un site distant, sont acceptées comme transactions locales, dés lors que le site distant est identifié.

8.2.3 L'intégrité sémantique

Le contrôle d'intégrité sémantique garantit la cohérence de la base de données distribuée, en rejetant les opérations de mises à jour qui produisent des états incohérents. La difficulté majeur posée par la distribution, concerne la définition de ces contraintes ainsi que leur vérification.

Pour résoudre ce problème, on utilise le site qui a initié la transaction comme *site maître,* et pour chaque contrainte portant sur une relation affectée par la mise à jour, le site maître exécute une requête qui retrouve les tuples qui ne la satisfont pas. Une relation résultat non vide, indique la violation de l'intégrité sémantique. La transaction est alors rejetée avec renvoie de messages d'erreurs à l'utilisateur final. ***[GAR 91]***

8.3 Evaluation des requêtes distribuées

L'évaluation des requêtes dans une base de données distribuée, désigne la fonction qui transforme une requête distribuée en un plan d'exécution distribué. Celui ci est un ensemble d'opérations de traitements de données locales, et d'opérations de communication de données intermédiaires ***[GAR 91].***

Dans un système de gestion de base de données centralisée , l'évaluation de requête est divisée en trois phases successives : (figure 2.7)***[BOU 03]***

- *Décomposition :* durant cette étape, la requête SQL est transformée en arbre algébrique.
- *Optimisation* : des optimisations sont appliquées sur l' arbre issu de la phase précédente, afin de déterminer l'ordonnancement optimal des opérations relationnelles, ainsi que les algorithmes d'accès aux données. L'optimisation utilise :

    - Les propriétés des opérateurs algébriques (Associativité, Commutativité, Distributivité)
    - Des heuristiques : en évaluant en premier les opérations les moins coûteuses, par exemple en faisant descendre *les sélections,* ce qui réduit le nombre de tuples, ainsi que *les Projections* qui réduit le nombre de colonnes.

- *Exécution* : c'est l'étape finale, qui consiste dans l'exécution de la requête optimisée afin d'élaborer son résultat.

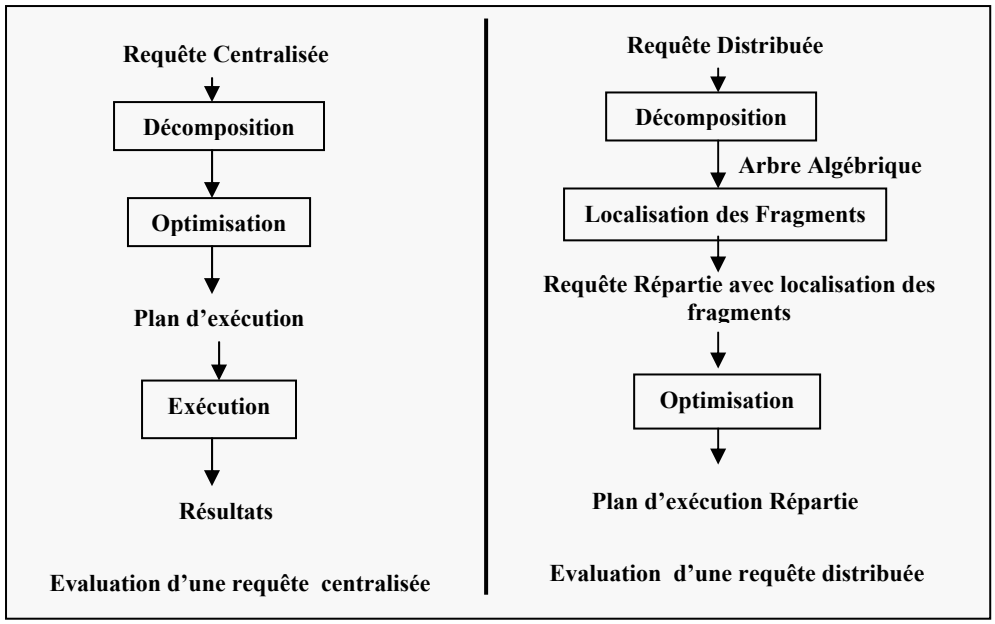

FIG. 2.7 - Evaluation d'une requête centralisée versus distribuée

L'évaluation d'une requête dans une base de données distribuée, introduit une phase supplémentaire : *la localisation* (Figure 2.7)**.** Dans un SGBD distribué, les étapes suivantes sont appliquées pour le traitement d'une requête :***[BOU 03]***

- La décomposition de la requête distribuée.
- La localisation des fragments.
- L'optimisation.
- L'exécution.

Dans ce qui suit, nous détaillons les différentes étapes d'évaluation des requêtes distribuées :

8.3.1 La Décomposition

Durant cette étape, le SGBD distribué décompose la requête écrite dans un langage de haut niveau (SQL) en arbre algébrique ; en étendant cette représentation, par *les sous arbres de reconstructions des fragments.**[BOU 03]*** qui sont des arbres décrivant les requêtes de reconstructions des fragments.

Exemple : dans la figure 2.8, on illustre l'arbre algébrique de la requête *Req* ainsi que les sous arbres de reconstructions des fragments***. [BOU 03] [BOU 02b]***

Relation : *Produit* (Prod, Libellé, Pu…,) ; *Stock* (Dépôt,…)
La requête globale *Req* = Quels sont les produits du dépôt n°4 ?

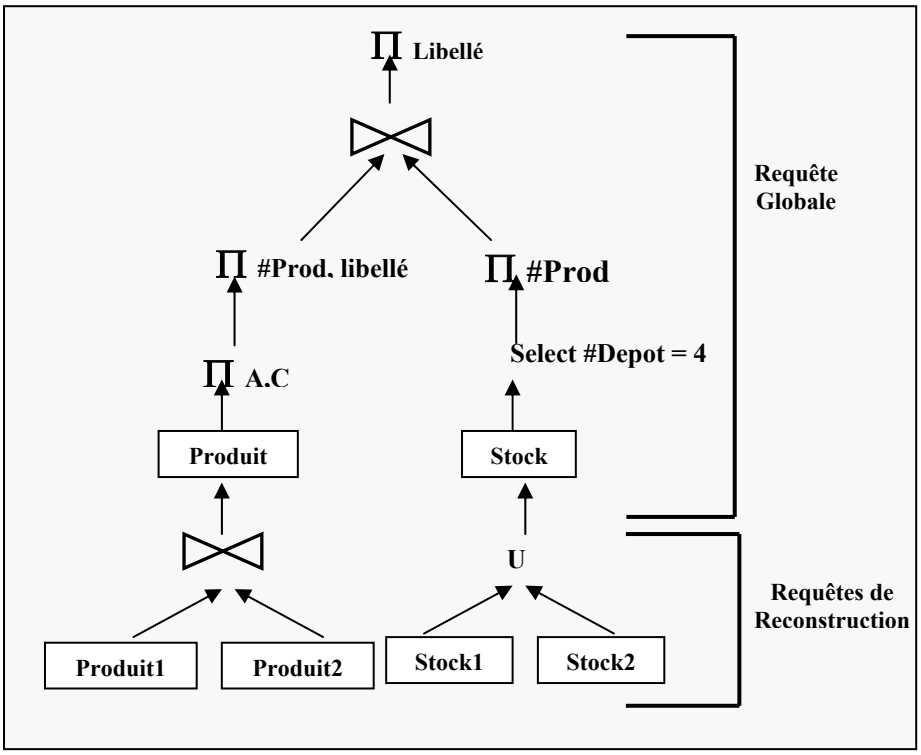

FIG. 2.8 - Arbre algébrique de la requête et ses sous arbres de reconstructions

Tels que les fragments *Produit1* et *Produit2* de la relation *Produit, Stock1* et *Stock2* de la relation *Stock* sont décrits par la figure 2.9 :

= **Proj** #prod,libellé (Produit)
Produit2 = **Proj** #prod,Pu (Produit)

Stock1 = **Proj** # Dep=4 (Stock)
Stock2 = **Proj** # Dep <> 4 (Stock)

FIG. 2.9 – Fragments des relations *Produit et Stock*

8.3.2 La localisation

L'opération de localisation, consiste dans la définition de la fragmentation des relations***[GAR 91].*** C'est le schéma de placement qui contient toutes les informations nécessaires à la localisation des fragments d'une relation. A partir des informations des fragments et copies de la base, le SGBD distribué, augmente la requête avec les informations de localisation des différents fragments et copies dont elle fait référence.

La localisation d'une requête distribuée procède en deux étapes ***[GAR 91]*** **:**

- *Génération de requête canonique :* la requête canonique est obtenue en remplaçant chaque relation de la requête distribuée globale par la requête de reconstruction correspondante (figure 2.8)

- *La simplification :* elle est basée sur des règles de simplifications, qui permettent de déterminer les opérations inutiles dans une requête canonique. Les opérations inutiles sont celles qui produisent un résultat vide, ou identique à l'opérande. Parmi ces règles, on cite :

    1. Une opération de restriction sur un fragment horizontal, dont le prédicat est en contradiction avec le prédicat de fragmentation, produit un résultat vide.
    2. Eliminer les conditions de sélections *inutiles,* quand la condition est identique à celle de la fragmentation.
    3. Une opération de projection sur un fragment vertical, dont tous les attributs projetés- à l'exception, de l'attribut commun de reconstruction- n'appartiennent pas au fragment, donne un résultat vide .

Ces règles de simplifications procèdent à une optimisation préliminaire de la requête canonique, par la suite, dans l'étape de l'optimisation, des opérations plus complexes seront détaillées.

8.3.3 L'optimisation

Le rôle de l'optimisation, est de déterminer une stratégie d'exécution de la requête distribuée qui minimise une fonction de coût, en évitant ainsi des stratégies qui conduisent à de mauvaises performances. Les fonctions de coût à minimiser sont généralement les suivantes : ***[GAR 91]***

- *Le temps total d'exécution* : Dans le cas d'un SGBD centralisé, ce temps est la somme, du temps des entrées-sorties (accès disque) et du temps du calcul en unité central. Dans le cas distribué, on ajoute le temps de communication nécessaire, à l'échange des données entre différents sites participants à la requête distribuée. Minimiser le coût de cette fonction augmente le débit du système (nombre de transactions par seconde).
- *Le temps de réponse :* Il prend en compte les traitements exécutés en parallèle.

Pour exécuter une requête, on peut utiliser par exemple la stratégie simple suivante : exécuter les opérations unaires sur les fragments comme des requêtes locales, puis transférer les résultats vers un site unique afin d'y assembler le résultat final. ***[GAR 91].*** L'optimisation doit considérer différentes stratégies d'exécution, et prédire leur coût d'exécution, afin de générer *le plan d'exécution.*

Le plan de l'exécution est l'ensemble des traitements locaux, ainsi que les opérations de communications des données intermédiaires, nécessaires pour les exécutions locales et la synchronisation globale. ***[BOU 02a] .*** Parmi les décisions incluses dans un plan d'exécution , on peut trouver :

- *Ordonnancement optimal des jointures :* choix de l'approche de réalisation de la jointure résidant à des sites différents, qui peut être réalisée de manière directe en transférant l'une des relations sur le site de l'autre où de manière indirecte en remplaçant la jointure par une combinaison de semi-jointures ***[GAR 91].***

- *La sélection des sites d'exécution :* selon le coût des communications, qui doit être minimum.
- *La sélection des copies :* Site le plus proche, le moins engorgé.
- *Le choix des algorithmes d'accès distribué*
- *Les algorithmes d'accès locaux*
- *Mode de transfert de données inter-sites :* exemple un tuple à la fois ou un ensemble à la fois.

L'un des facteurs décisif, dans le choix de la stratégie d'exécution *(la plus optimale),* est la taille des résultats intermédiaires. En effet, pour estimer le coût d'une stratégie, il est impératif de pouvoir estimer la taille de ces résultats. Pour cela ,on utilise les statistiques d'un fragment (*profil de fragment*) qui comportent des informations sur le taille de celui ci, sur ses attributs…etc. ***[GAR 91]***

8.3.4 L'exécution

Lors de l'exécution, on envoie la requête aux sites impliqués. Comme dans un traitement centralisé, dans chacun d'eux on applique une :

- Une optimisation locale.
- Une évaluation de la requête.
- Une réponse sera envoyée au site émetteur.

8.4 Gestion des transactions

Une transaction est une séquence de traitement atomique. Appliquée à une base de données cohérente, une transaction restitue une base de données cohérente***.[NB03] [GAR 91].*** Parmi les actions clés d'une transaction on peut citer *la validation* , *l'annulation ;* lors de l'échec d'exécution d'une transaction, un mécanisme de *reprise* assure la cohérence de la base de données. ***[GAR 91]***

Une transaction distribuée est constituée de plusieurs sous transactions, chacune s'exécutant à un site différent. Il est donc nécessaire, de gérer une information globale qui permet de décider de la validation ou l'abandon des mises à jour locales. ***[GAR 91]***

8.4.1 Propriétés ACID des Transactions

Une transaction est caractérisée par les quatre propriétés suivantes : ***[STN 98]***

- ***A**tomicité* :La séquence d'actions d'une transaction est indivisible. Toutes ses actions doivent être effectuées pour la *valider*, sinon la transaction est *annulée*
- ***C**ohérence* : L'exécution d'une transaction dans un environnement sans pannes préserve la cohérence de la base de données**.**
- ***I**solation* :Les actions d'une transaction sont isolées, ses résultats intermédiaires sont masqués aux autre transactions.
- ***D**urabilité* :On ne peut annuler les résultats d'une transaction qui s'est bien terminée.

8.4.2 Modèle de système de gestion de transactions

Dans un environnement de bases de données distribuées, l'exécution d'une transaction implique plusieurs sites. De ce fait, les opérations de *validation/annulation* (Commit/Rollbak) doivent être exécutées sur chacun des sites. Un modèle de système de gestion de transaction distribué est alors utilisé, pour coordonner entre toutes ces opérations. Il se compose de : ***[BOU 02a]***

- *Un gestionnaire local de transaction par site :* Il gère le log (*l'historique des transactions)*. Il participe aussi à la coordination des exécutions parallèles.

- *Un coordinateur de transactions :*Le coordinateur lance l'exécution d'une transaction, il la décompose en sous transactions en fonction de la localisation. Puis, il transfère les sous transactions vers les sites dédiés et coordonne leur validation.

8.4.3 Déroulement d'une transaction dans un SGBD distribué

L'étape la plus importante dans le gestion d'une transaction dans un contexte distribué, est celle de *la validation.* Elle consiste dans l'intégration définitive des mises à jour dans la base de données. Le protocole le plus utilisé, pour assurer l'atomicité d'une transaction globale, est *le protocole de validation en deux étapes* (*Two Steps Commit Protocol)* proposé par *Lampson et Gray****[GAR 91]*** .Ce protocole, est exécuté sur chaque site ; il est contrôlé par le site maître (*Coordinateur*). Comme son nom l'indique, il repose sur deux phases : (figure 2.10).

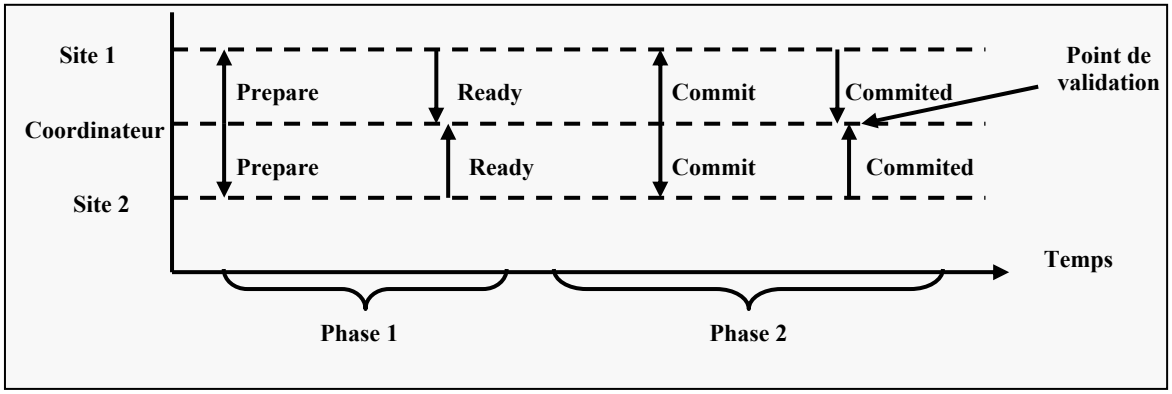

FIG. 2.10 - Validation Réussite

- *La préparation :* Le coordinateur demande aux sites concernés par la transaction, de se préparer pour la validation.
- *La validation :* le Coordinateur ordonne -aux sites participants à la transaction- la validation de leurs mises à jour ,s'ils ont tous complété la première étape, ou sinon les abandonner.

Les cas de pannes se présentent sous trois formes : ***[BOU 94]***

- P*anne du coordinateur* lui même : dans ce cas, les sites participants qui se sont déclarés prêts à valider sont bloqués, d'où le blocage des ressources jusqu'à la reprise de coordinateur.
- *Panne d'un des sites* ou *rupture de liaison* (figure 2.11) : ces deux cas sont traités de manière similaire, si les accusés *Ready* des sites ne parviennent pas au coordinateur dans un délai précis ; celui ci considère que la réponse est négative et leur envoie une annulation (*Abort).*

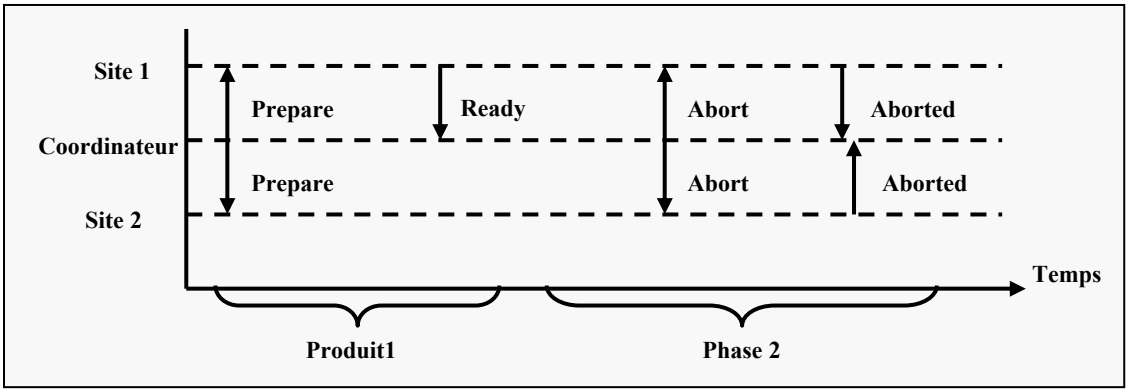

FIG. 2.11 - Panne d'un site

Quand le site *coordinateur* est en panne, les participants prêts à valider sont bloqués, on dit alors que le protocole *a deux étapes* est *bloquant* ***[GAR 91] [BOU 94].*** Afin de résoudre ce problème, le protocole à deux étapes est augmenté d'une nouvelle étape de préparation, et appelé *protocole de validation en trois étapes*. On rajoute entre l'état « prêt » et l'état « valider » un état appelé « prêt à valider ». Si le coordinateur est en panne, et qu'aucun participant n'a reçu le message « prêt à valider », alors la sous transaction est abandonnée afin de libérer les ressources***. [GAR 91],[BOU 02a] [BOU 02b]***

8.4.4 Gestion des accès concurrents

Deux transactions qui s'exécutent en parallèle, entrent en conflit si elles opèrent sur les mêmes données, et qu'au moins l'une d'elle, opère en écriture. Les problèmes de concurrences peuvent entraîner des : ***[GAR 91]***

- Pertes de mise à jour.
- Inconsistance de mise à jour.

Le contrôle de concurrence repose sur le principe de *sérialisabilité* : il garantit que, l'exécution parallèle des deux transactions est permise, si et seulement si elle est équivalente à

une exécution séquentielle de ces deux transactions (en série) ***[BOU 03].*** Les plus importantes techniques utilisées dans la gestion des transactions concurrentes consistent dans : ***[BOU 03]***

- *Technique d'estampillage (Time Stamping)* **:** Dans cette technique, l'ordre de la sérialisation est obtenu par association d'une *estampille* unique à chaque transaction. Chaque donnée garde trace de la dernière transaction qui l'a utilisée. L'ordonnancement suit l'algorithme suivant (figure 2.12):

Une Transaction Ti accède à une donnée estampillée j
Si j < i **alors** Ti peut s'exécuter
Sinon ABORT(Ti) et Reprise ultérieure avec estampille > j**.**

FIG. 2.12 - Algorithme de *Time Stamping*

Les estampilles peuvent être gérées, soit de manière centralisée ou distribuée. Dans le premier cas , un site central utilise un compteur logique ou une horloge locale pour créer et attribuer les estampilles. Dans le deuxième cas, les estampilles sont gérées sur chaque site, une synchronisation des horloges et compteurs est nécessaire.

- *Technique de verrouillage (Two phase locking)* **:** Contrairement à la technique d'estampillage, celle ci est implantée. Le verrouillage évite les conflits en suspendant au plus tôt, la progression des transactions qui tentent d'exécuter des opérations conflictuelles sur un même granule ***[GAR 91].*** Cet algorithme se compose de deux phases : *acquisition de verrous* , *libération sans nouvelle acquisition.* Cette technique est utilisée avec une gestion d'inter blocage. Dans un contexte distribué, plusieurs stratégies existent pour réaliser le verrouillage, soit en utilisant un *gestionnaire de verrous GV* par site, ou un seul *coordinateur Sc*.

L'interblocage peut être évité par l'utilisation *d'un graphe d'attente global* (figure 2.13), qui représente les transactions qui s'attendent. Ce graphe est réalisé par scrutations périodiques des transactions. Il peut être géré de manière centralisée, dans ce cas , quand un cycle existe, une transaction est tuée. Dans une gestion distribuée, un graphe par site est crée, un graphe global avec inter blocage (*Deadlocks)* est alors obtenu. ***[GAR 91].***

Dans le figure 2.13 , la notation « T1→ T2 » indique que la transaction T1 attend la libération d'un verrou obtenu par T2. On constate dans le graphe global, que les transactions T2,T3 et T4 sont en situation de *verrou mortel global (deadlock)* , la résolution de ce verrou mortel consiste à éliminer le circuit, en abandonnant l'une des transactions impliquées. La transaction tuée est généralement celle qui a fait le moins de travail (la plus jeune, ou celle qui a obtenu le moins de verrous) ***[GAR 91].***

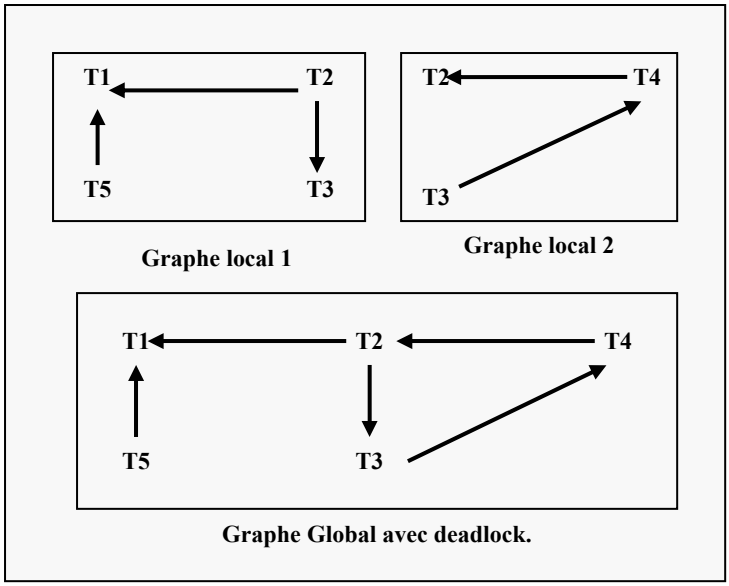

FIG. 2.13 - Interblocage : graphes locaux et graphe global

8.5 Sécurité de fonctionnement

Le système de gestion de base de données distribué doit assurer la sécurité de fonctionnement des opérations, et par conséquent, la reprise en cas de pannes. Les types de pannes qui peuvent apparaître dans de tels systèmes, sont résumés dans ce qui suit***[BOU 03]*** *:*

- Panne des sites.
- Rupture d'une liaison.
- Perte de messages.
- Partition du réseau : une partie du réseau reste disponible alors que l'autre est en panne.

La procédure de reprise en cas de pannes se compose des étapes suivantes :

- *La Détection de la panne* : La détection de la panne est généralement possible, cependant l'identification de celle ci n'est pas évidente. Dans le cas où deux sites ne communiquent plus par exemple, on peut supposer que ,soit il y'a eu rupture de liaison, ou que l'un des deux sites est en panne.

- *La reconfiguration du système :* Après la détection de la panne, il est question de reconfigurer le système, afin qu'il puisse continuer à travailler. La reconfiguration peut se résumer dans les actions suivantes ***[BOU 03] :***
  - Avortement (annulation) des transactions actives.
  - Inhiber l'accès aux données dupliquées du site en panne.
  - Si le site coordinateur est en panne, il faut procéder à son remplacement.

- *La Reprise après pannes* **:**Apres la reconfiguration du système, il faut appliquer les mises à jour qui ont eu lieu sur les données dupliquées. Pour éviter les risques de blocage, on duplique le site coordinateur. Plusieurs stratégies de duplication existent :

- *Utiliser un coordinateur Back up* : il reçoit les mêmes messages que le vrai coordinateur, exécute les mêmes algorithmes, et prend le relaie en cas de panne.
- *Elire un coordinateur :* quand une panne arrive, on choisit un site coordinateur et on informe les autres sites.

9. Les Bases de données fédérées (MultiBases de données)

Les bases de données distribuées présentent l'inconvénient de l'administration centralisée de la base, qui peut s'avérer coûteux et complexe, voire impossible, en présence d'un grand nombre de bases de données locales. De plus, les accès de l'utilisateur à sa base de données locale sont ralentis par les coûts supplémentaires de gestion des données réparties***[GAR 91]***. Pour résoudre ces problèmes, on fournit à l'utilisateur un langage pour définir les relations entre différentes bases, c'est les *bases de données fédérées (Multibases de données)*. ***[BUK 96]***

Une MultiBase de données *(bases de données fédérée)* consiste dans un système distribué, qui fournit une interface globale à plusieurs SGBDs locaux, hétérogènes et pré-existants, qui ne sont pas intégrés en une seule base ***[BUK 96] [GAR 91]***. De tels systèmes introduisent les nouvelles fonctionnalités suivantes : ***[BUK 96]***

- Fonction de traducteur de modèle de données.
- Fonction de traducteur des langages de manipulation .

L'approche d'intégration des données choisie dans une base de données fédérée, est celle du *Modèle Pivot* ***[BOU 95]*** (figure 2.14). En effet, un modèle de données commun pour la description des schémas au niveau global, ainsi que le langage de manipulation de données (*LMD)* qui lui est associé, sont utilisés par le système. Celui ci prend en charge de manière automatique et transparente, les traductions des modèles de données et langages d'accès locaux vers le modèle\langage pivot. Il est à noter, qu'il existe autant de traducteurs, que de modèles locaux***.[BUK 96]***

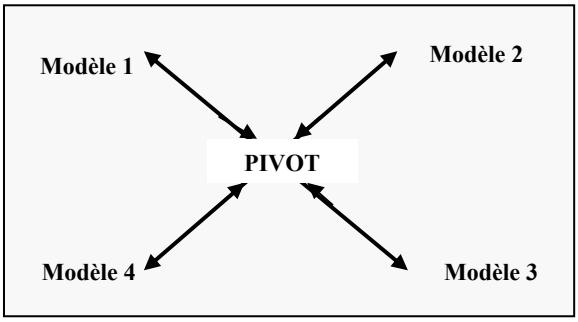

Fig. 2.14 - Intégration - approche pivot

Chaque SGBD local *participe* dans le système global, en partageant une partie ou la totalité de sa base, les données partagées sont ainsi définies dans une vue présentée à l'interface utilisateur du SGBD local. En résumé, un SGBD fédéré diffère d'un SGBD réparti, par les points suivants : ***[GAR 91]***

- Il ne gère pas de dictionnaire de données global.

- Il supporte un langage pour définir les dépendances qui peuvent exister entre différentes bases de données.
- Il supporte un langage pour définir et manipuler les bases de données appartenant à la fédération.

9.1 Architecture à 5 niveaux de Sheth et Larson

Sheth et *Larson* ont proposé une architecture générale des bases de données fédérées, de la façon qui suit (figure 2.15) : ***[BUK 96]***

- Niveau 1, *le schéma interne local* : Chaque SGBD local a un schéma qui définit la totalité des données locales.
- Niveau 2, *le schéma des composants (Schéma conceptuel local)* **:** Ce niveau représente Le schéma *des composants,* qui est le schéma local représenté dans le modèle de données global.
- Niveau 3, *le schéma d'exportation (schéma externe local)* : C'est le schéma d'exportation, qui est une représentation , dans le modèle de données global ,de la partie du schéma local qui va être partagée. Il peut exister plusieurs schémas d'exportation de chaque SGBD local.
- Niveau 4, *le schéma fédéré (schéma conceptuel global) :* il est formé d'une combinaison de schémas d'exportations. Les schémas fédérés représentent des portions du schéma global logique.
- Niveau 5, *le schéma externe (global) :* Des schémas externes sont définis sur les schémas fédérés ; grâce auxquels, les utilisateurs peuvent accéder aux schémas fédérés.

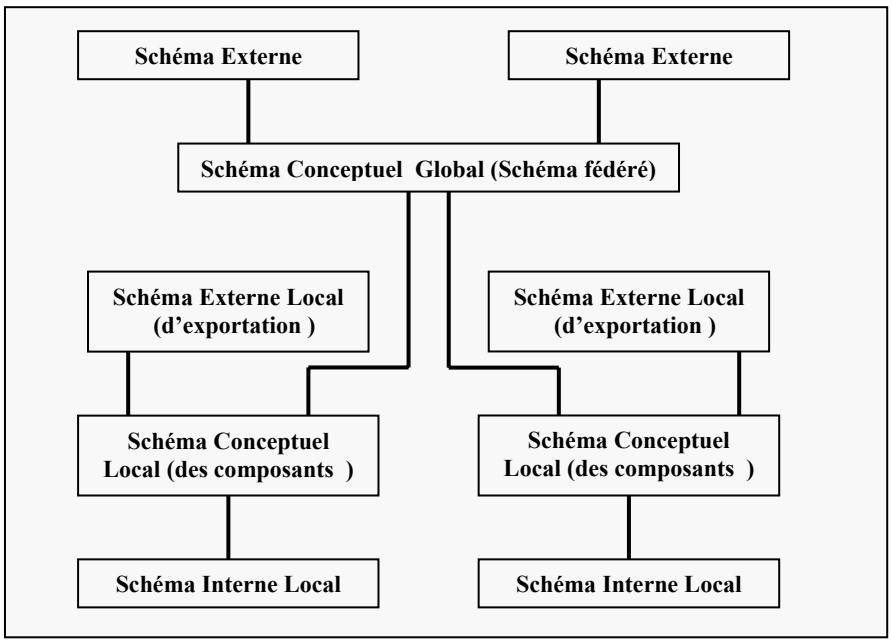

FIG. 2. 15 - Modèle de base de données fédérée avec schéma conceptuel global

Les plus répandues des implantations de cette architecture sont :

- *Base de données fédérées avec schéma conceptuel global* ***[BUK 96]*** (figure 2.15). L'existence de *schéma conceptuel global* permet, de maintenir les différents schémas externes de manière centralisée.

- *Base de données fédérées sans schéma global* ***[GAR 91]***: la motivation essentielle de cette approche, est d'assurer l'autonomie totale des différentes bases de données ; ce qui permet d'effectuer les différentes opérations d'administration et de gestion de manière indépendante. L'absence de schéma global permet de tolérer les conflits sémantiques, qui posent problème à la conception.

10. Conclusion

Les bases de données distribuées ont été les premières tentatives de l'industrie Informatique, pour fournir une interface unique et unifiée aux différentes données, quels que soient leurs formats et emplacements. Appuyé sur le modèle relationnel, on a présenté dans ce chapitre, les concepts de base, de ces systèmes. Des techniques de distribution de données, d'exécution de requêtes distribuées ont été présentées .On a décrit également, la gestion des transactions, ainsi que la gestion des pannes. Nous avons terminé le chapitre par une étude sur les systèmes de bases de données fédérées ; on a entre autre expliquer les points de différences qu'ils présentent avec les systèmes de gestion de bases de données distribuées.

En utilisant ces mécanismes , les SGBDs fédérées ont tenté de répondre aux besoins de la distribution. Cependant, ils ont vite été abandonnés à peine expérimentés, à cause de la complexité de leur mise en œuvre, de leur coût de déploiement et surtout des performances insignifiantes des requêtes distribuées. En effet, le modèle relationnel n'est pas suffisamment puissant pour répondre aux exigences de tels systèmes.

Chapitre 3 : Approches d'intégration.

1. Introduction

Des nouvelles solutions informatiques pour les entreprises apparaissent chaque jour ; des solutions hétérogènes, et qui doivent pourtant collaborer, facilitant ainsi aux entreprises le déroulement de leurs processus métier. Cependant, la rude concurrence que celles ci subissent aujourd'hui, ainsi que les exigences qualité/prix, ont poussé les dirigeants informatiques à déployer des solutions qui permettent l'intégration de ces applications hétérogènes, afin de pouvoir répondre en temps réel, et à moindre coût, aux exigences de leurs clients ***[FOU 01]***. En effet, une intégration de l'ensemble des ressources de l'entreprise s'avère nécessaire, non seulement pour la prise en compte de l'existant, mais aussi pour assurer l'exploitation de la connaissance contenue dans ces diverses sources d'informations.

Les bases de données distribuées, étudiées dans le chapitre précédent, ont mis les premières briques de solutions pour ce problème. Cependant, en pratique, les objectifs que ces systèmes voulaient atteindre n'étaient par réalisables. Parti de ce constat, un ensemble d'outils, destinés à l'entreprise ont été développé ; ils présentent la particularité de répondre qu'à un sous ensemble spécifique des exigences des systèmes distribués.

Dans ce chapitre , nous mettons l'accent sur ces approches. Apres « l'intégration point à point » décrite dans la section (2) ; on présente l'EAI (*Entreprise Application Integration*) , dans la section (3). Les services web sont détaillés dans la section (4), les entrepôts de données en section (5). Une synthèse de ces trois approches ainsi qu'une étude comparative avec les bases de données fédérées sont présentés dans la section (6). On conclut dans la section (7).

2. Intégration « Point à Point »

Dans cette approche, chaque source de données est intégrée en utilisant un format de données spécifique, et le protocole d'échange qui lui est associé. Pour chaque intégration de ce type, on procède à des opérations de paramétrages et d'optimisations spécifiques.***[AVI 99]***

Quand deux applications communiquent (figure 3.1) , le mécanisme mis en œuvre consiste à extraire une partie de l'information contenue dans l'application, puis, pour chaque destinataire, la formater et la transmettre. L'extraction ainsi que le formatage des données constituent généralement des développements spécifiques ; quant au transport des données, il est assuré par des transferts "classiques" (FTP, partage de fichiers) .

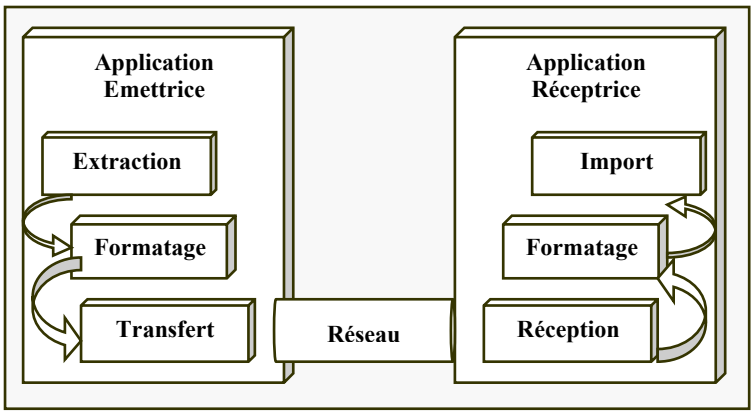

FIG. 3.1 – Intégration « Point à Point

3. L'intégration par «Hub de services» : EAI

L'EAI (*Entreprise Application Intégration*) est l'ensemble des produits d'intégration d'applications en entreprise. Il permet d'échanger de manière performante des informations entre applications ou progiciels s'exécutant sur des plates-formes hétérogènes, dans des Systèmes d'Informations en constante évolution. L 'EAI représente la seconde génération des outils d'intégration. ***[GAR 03a] [AVI 99]*** **.**

L'EAI représente l'intégration intra-entreprise , qui est aussi désigné par le sigle A2A (Application to Application), c'est une intégration qui se limite aux applications internes à l'entreprise. ***[MED 04]***

3.1 Principe de Base

L'intégration consiste à fédérer les échanges d'information autour d'un Bus d'Echange (*Hub*). En effet, le hub centralise les flux, en assurant les transformations et les routages nécessaires. Il fonctionne en mode "*Publish and Subscribe*"(figure 3.2) : dès qu'une information présente dans une application est susceptible "d'intéresser" d'autres applications, elle est transmise, non pas aux destinataires finaux, mais au bus d'échange. Celui-ci a la charge d'en assurer ensuite le routage vers les applications "intéressées", en y appliquant éventuellement les transformations nécessaires à leur bonne interprétation.

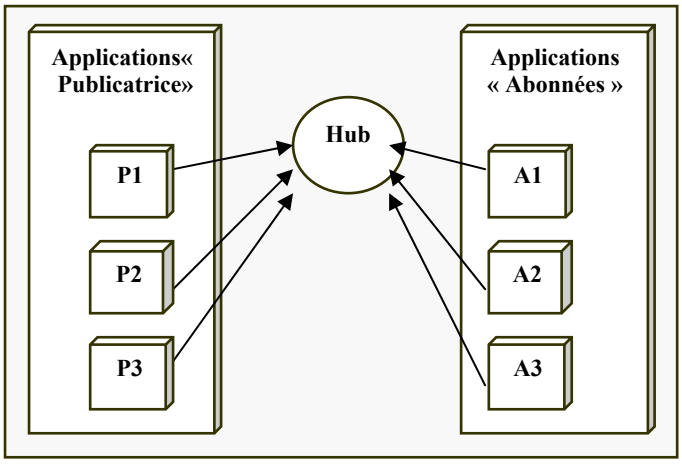

FIG. 3. 2 - Principe du Hub Central

3.2 L'architecture d'un EAI

3.2.1 Mécanismes d'échange

Dans une architecture EAI, le mécanisme d'échange d'information entre applications est basé sur des messages qui incluent les données, et les méthodes qui leurs sont associées. Exemple : envoi d'un message *Commander* contenant la quantité, le code de l'article ; envoi d'un message *Régler* contenant la devise, le client et le montant, etc. Sur le marché, il existe deux technologies implémentant l'échange de messages entre applications : ***[AVI 99]***

- *Les Object Request Broker (ORB):* Ce type d'architecture est basé sur des messages envoyés en *mode synchrone.* L 'émetteur doit localiser le destinataire sur le réseau, se connecter à l'une de ses instances et attendre la réponse. Dans un tel mode, la requête

n'est jamais perdue. En cas d'erreur ou de rejet, l'application émettrice en est directement avertie. Les technologies *RPC, HTTP, CORBA* ***[BOU 96]*** *,DCOM,* IIOP, RMI, …etc. entrent dans cette catégorie. L'un des inconvénients de l'utilisation d'ORBs , est qu'ils contraignent les applications à dépendre les unes des autres.

- *Les messages orientés MiddleWare (MOM) :* A l'inverse des *ORBs*, les *MOMs* utilisent le *mode asynchrone,* où la disponibilité du destinataire au moment de l'émission n'est plus indispensable ; celui-ci traitera la requête en temps voulu. Charge au MiddleWare utilisé de garantir alors, que la requête ne se perde pas. Les technologies *MQSeries de IBM , Microsoft MSMQ, Tibco/Rendez-Vous* entrent dans cette catégorie.

Seules les MOMs sont aujourd'hui capables d'implémenter les fonctions de transformation et de contrôle de flux dont un système EAI à besoin. Cependant , il est des cas où le mode synchrone question/réponse est nécessaire. ***[AVI 99]***

3.2.2 Echange en mode Message MOM

Dans un scénario MOM , une application émettrice poste un message dans une file d'attente précise. L'application réceptrice, quand elle le décide, consomme les messages postés par le programme émetteur. Les deux applications communiquent donc directement, au travers d'une file d'attente commune des deux parties (figure 3.3). ***[AVI 99]***

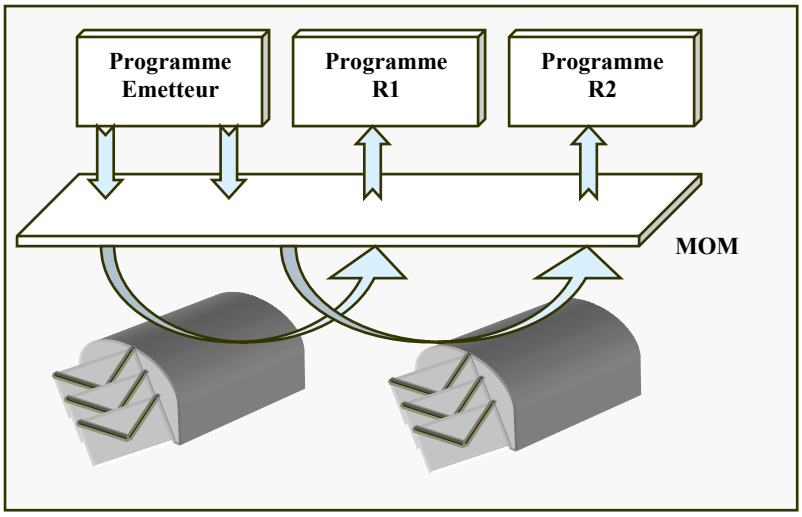

FIG. 3.3 - Echange via un MOM

Dans une architecture EAI, les MOMs traditionnels sont augmentés d'une fonctionnalité importante le *Publish-and-Subscribe* : où l'échange n'est plus 1 à 1, mais 1 vers N. Un message est envoyé par un émetteur à plusieurs destinataires. Actuellement on peut réaliser deux modes de *Publish- and- Subscribe* : ***[AVI 99]***

- *Mode Décentralisé :* Le principe est simple, les applications s'abonnent en utilisant des verbes de publication : *mq_put(domaine_de_publication, message)*, les applications réceptrices s'abonnent avec des verbes de suscription: *mq_subscribe(domaine_de_publication)* .

- *Mode Centralisé :* Dans ce mode, c'est un *message Broker* sous forme de *Hub* centralisé, et par lequel passent tous les messages des applications (messages de publication et de souscription). Les applications utilisent toujours l'API *mq_put/mq_get* du MOM ,comme dans le cas du point à point, mais uniquement pour communiquer avec le Hub : N applications génèrent N flux possibles (configuration en étoile vers le Hub).

Alors que *le Publish/Subscribe décentralisé* s'applique à l'échelle d'une application, le *Publish/Subscribe centralisé*, lui s'adapte à une architecture EAI, à l'aide d'un Message Broker, il va permettre d'assurer au niveau central ,un certain nombre de services additionnels qui vont permettre de diminuer la part de code nécessaire à la communication dans les applications.

3.2.3 Les Fonctionnalités du MOM

Au-delà de la simple gestion des files d'attente, le MOM assure un certain nombre de fonctionnalités pour sécuriser l'architecture : ***[AVI 99]***

- *Garantie de délivrance :* Un message, une fois soumis par l'application, est soit consommé par l'application réceptrice ou si sa durée de vie expire, envoyé dans une file d'attente particulière .

- *Mode Synchrone:* On simule ce mode, en utilisant une file *Reply queue* ,dans laquelle l'application émettrice attend une réponse en retour de son message.

- *Priorité, groupage des messages :* En marquant les messages comme ayant une priorité particulière.

- *La sécurité :* Un message peut être authentifié, son contenu rendu intègre ou confidentiel. De même, pour une file d'attente, dont on peut restreindre l'accès. Par exemple tout le monde ne doit pas pouvoir publier des messages dans une file d'attente "paiement" sans contrôle.

- *Triggering :* Sur l'arrivée d'un message dans une file d'attente, il est possible de déclencher automatiquement un programme. Par exemple, une transaction, une mise à jour des bases de données, …etc.

3.3 Les Fonctions d'un EAI

Les fonctions principales d'un EAI peuvent se résumer dans ce qui suit (figure 3.4):

- Transformation de données.
- Interfaçage et passerelles techniques entre applications .
- Gestion des processus.

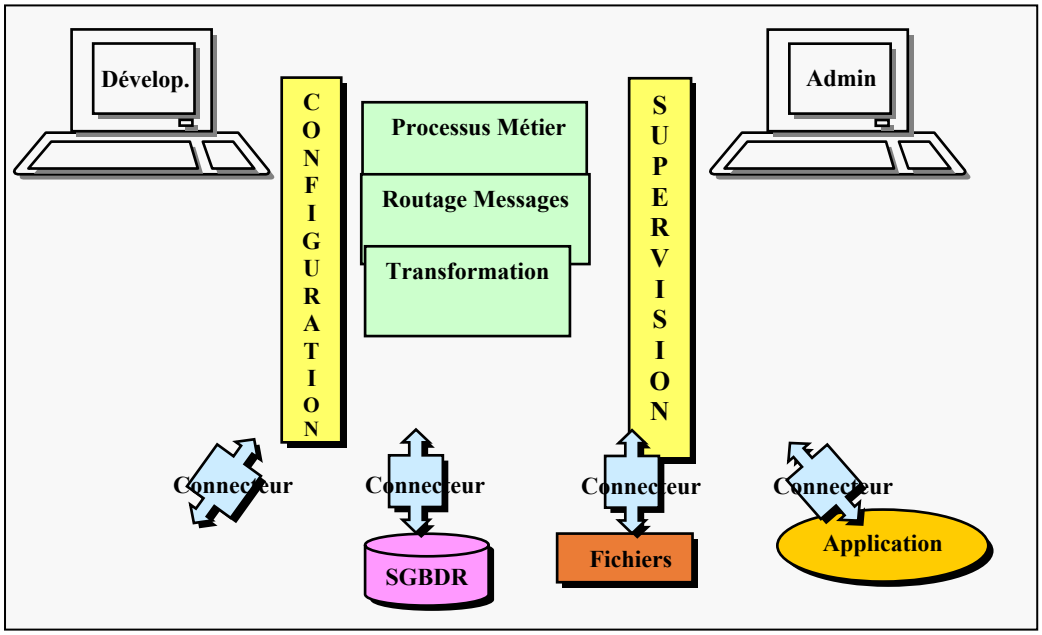

FIG. 3.4 – Architecture Fonctionnelle d'un EAI

3.3.1 Transformation de données

Le formatage de données consiste dans la transformation des données dans un format compréhensible par les applications intégrées par l'EAI ***[GAR 03a]*** **.** Dans la communication entre deux systèmes, la phases de *formatage des données* envoyées est primordiale, puisque elle permet aux applications de comprendre, le langage des unes des autres (figure 3.4).

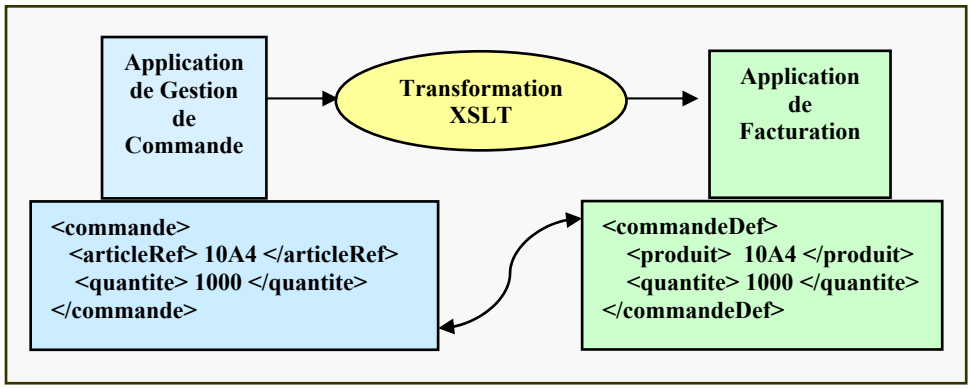

FIG. 3.5 – Phase formatage de données

3.3.2 Interfaçage et Passerelles techniques

L'architecture EAI repose essentiellement sur ses possibilités d'interfaçage avec les systèmes existants. Les MOMs et les ORBs proposent en standard des interfaces vers : ***[AVI 99]***

- *Des SGBDs relationnels* **:** l'EAI permet de se connecter à des sources de données relationnelles, afin de mettre à jour des données sur arrivée de message, d'envoyer des messages sur modification de données. (figure 3.5).

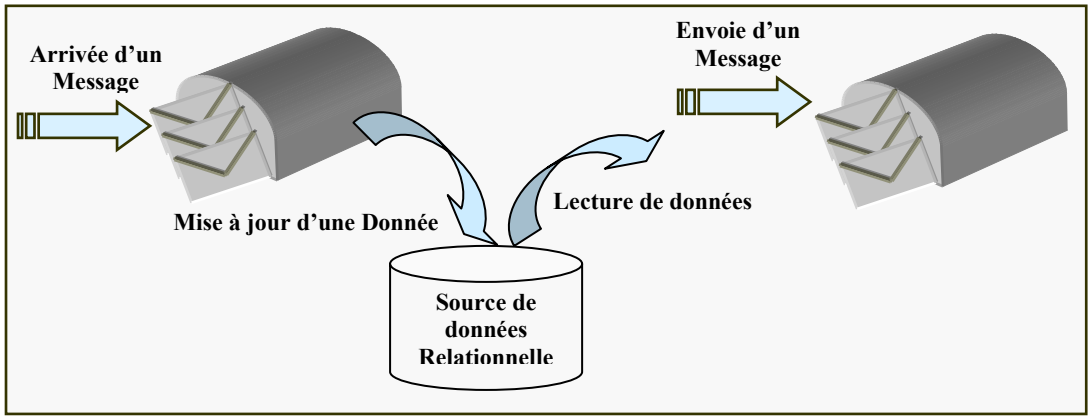

FIG. 3.6 - Passerelle avec les SGBDR

- *L'interconnexion avec les Serveurs d'applications :* l'EAI permet aussi d'interconnecter des composants. On peut alors envoyer des messages, depuis une transaction ou un composant, de déclencher une transaction ou une méthode de composant à, l'arrivée d'un message…etc.

- *Le transfert de fichiers :* L'interfaçage d'un EAI avec les mécanismes de transferts de fichiers est fondamentale, en particulier pour permettre une reprise de l'existant par étape. Les processus de transferts de fichiers pré existants, seront petit à petit intégrés dans le Bus d'Echange.

3.3.3 La gestion de processus ou Workflow

La notion de gestion des processus métier permet d'orchestrer et organiser les enchaînements des tâches entre les applications. L'avantage réside dans l'automatisation complète de ce processus de définition de la meilleurs combinaison des tâches au sein de la chaîne de valeur de l'entreprise. ***[Med 04]***

Pour ce faire, les Outils EAI s'équipe le plus souvent d'un moteur de Workflow , qui permet de faire circuler des documents, selon des processus complexes mettant en jeu des personnes physiques. Ce moteur constituent des middlewares perfectionnés permettant de relier les applications au processus***[STO 01]***. Le moteur de Workflow devra s'appuyer sur le Message Broker (Hub) pour définir et modéliser les flux métiers, ainsi que l'échange des messages. ***[GAR 03a]***

3.4 Mécanismes d'intégration des applications

Dans ce qui suit , on présente les mécanismes mis en œuvre, pour intégrer différentes applications ; telles que les applications client/ serveur SGBDR, les applications 3-tiers (serveurs d'applications) et ERP, en les adaptant au mode message. ***[AVI 99]***

3.4.1 Les applications client/serveur

Une application client/serveur est une application dont le module client dialogue avec le module serveur SGBD ; en utilisant des requêtes SQL, ainsi que des procédures stockées, deux techniques peuvent être utilisées : ***[AVI 99]***

- *Technique Intrusive :* Elle utilise une interface de programmation pour accéder au MOM ; en intégrant dans les procédures stockées, des envoies et des réceptions de messages. Très simplement, le développeur utilisera les procédures *MQOPEN/ MQCLOSE* pour ouvrir et fermer une file d'attente *; MQPUT/MQGET* pour déposer et retirer des messages. La figure 3.7 illustre l'exemple de l'utilisation de ces APIs dans une procédure stockée Oracle.

- *Techniques non intrusives* : Elle consiste à utiliser un outil de réplication qui "surveille" certaines tables du SGBD ; et émet un événement dès qu'une mise à jour se produit. Cet événement peut être utilisé pour générer un message MOM (MQSeries, MSMQ, etc.), qui pourra être traité dans le cadre de l'architecture.

```
DECLARE
      Queue              PGM.MQOD@pg4mq;
      MsgDesc            PGM.MQMD@pg4mq;
      GetOptions         PGM.MQGMO@pg4mq;
      ObjectHandl        BINARY_INTEGER;
       Message           RAW(32757);

BEGIN

  -- Ouvre la file d'attente QUEUE en lecture.
   Queue.OBJECTNAME : = 'QUEUE' ;
  PGM.MQOPEN @ pg4mq(  queue, PGM_SUP.MQOO_INPUT_AS_Q_DEF,
        ObjectHandle) ;

  -- Récupérer un message dans la queue :
    PGM.MQGET@ pg4mq(  ObjectHandle, MsgDesc , getOptions, message ) ;


  -- Traitement du message
  ......

  -- Fermer la file d'attente
  PGM.MQCLOSE @ pg4mq (Objecthandl, PGM_SUP.MQCO_NONE);

END ;
```

FIG. 3.7 - Technique intrusive

3.4.2 Les applications 3-tiers

Dans ce type d'applications, on a à traiter avec des composants serveurs standards, les mécanismes d'intégration les plus adéquats , consistent dans : ***[AVI 99]***

- *Utilisation d'interfaces natives* : API propriétaires du type *mq_get, mq_put et mq_subscribe*, disponibles en C/C++, en Java, Cobol, etc.

- *Utilisation de composants d'accès* : développement à base de composants techniques qui permettent d'intégrer l'application au bus de message.

3.4.3 Les ERPs (Enterprise ressource planning)

La plupart des ERPs proposent des interfaces, capable de lire et de modifier les informations qui leur sont internes ; afin d'intégrer le progiciel au reste du Système d'Information. Pour intégrer un progiciel en mode fil de l'eau, celui-ci doit supporter la notion de "trigger". A l'aide de ces triggers, un événement survenu dans le progiciel pourra aisément déclencher l'émission d'un message. Cependant, tous les ERPs ne supportent pas ce type de fonctionnement ; alors la méthode la plus communément proposée par les éditeurs, reste le polling (vérification périodiques). ***[AVI 99]***

4. L'intégration par «bus de services» : Les Services Webs

Les Services Webs représentent un nouveau modèle de communication programme à programme, permettant d'intégrer différentes applications rapidement, facilement et à moindre coût***[ABI 02b] [ABI 02a]***. Ils sont basés sur des standards existants et émergeants tels que : *HTTP , XML, SOAP (Simple Object Access Protocol)* , *WSDL (Web Services Description Languages)* et *UDDI (Universal Description, Discovery and Integration)*. ***[GIR 01] [ABI 02b]***

Les services web consistent à utiliser un vocabulaire commun (WSDL) pour tous les traitements, et puis d'assurer le transport des messages par sérialisation XML (SOAP). Un service Web est accessible depuis un langage de programmation quelconque, et donc depuis une plate forme quelconque. il peut être utilisé pour exporter des fonctionnalités d'un système, et les rendre accessibles via des protocoles standards, au moyen de MiddleWare (CORBA, RMI,DCOM, EJB, etc.) ***[KRE 01].*** Une des applications les plus importantes des services web, et qui est particulièrement intéressante pour le contexte de notre thèse , c'est l'accès à distance aux bases de données ***[ABI 02b].***

4.1 Définition

Les services Web sont des composants logiciels englobant des fonctionnalités métier de l'entreprise, et accessibles via des protocoles standards du Web. ***[GAR 03a]***

Un service Web est une *interface* permettant de décrire une collection d'opérations accessibles via le réseau en utilisant des messages basés sur XML. ***[GIR 01]***

Un service web est décrit par un document , au format XML spécifique appelé le langage WSDL (Web Service Description Language). *La description du Service* précise les méthodes pouvant être invoquées, leurs signatures, et les points d'accès du service (URL, port, etc.). ***[KRE 01].***

4.2 Modèle d'architecture d'un Service Web

Cette architecture est basée sur l'interaction entre trois composants (figure 3.8): ***[GIR 01] [KRE 01]***

- *Fournisseur Des Services (Services Provider) :*est une application s'exécutant sur un serveur, et comportant un module *Service,* accessible par le réseau en XML. ***[GAR 03a]***

- *Annuaire des Services (Services Registry) :* C'est un serveur d' application, qui peut être installé au niveau de l'entreprise ou au niveau mondial ; il comporte les descriptions des services, publiés par le *fournisseur*.

- *Demandeur de Service (Services Requestor) :*Est une application cliente (un navigateur , un programme sans interface utilisateur ou un autre service Web) qui recherche un service dans l'annuaire, se lie à lui, puis initie une interaction avec lui en invoquant ses fonctions par des messages XML.

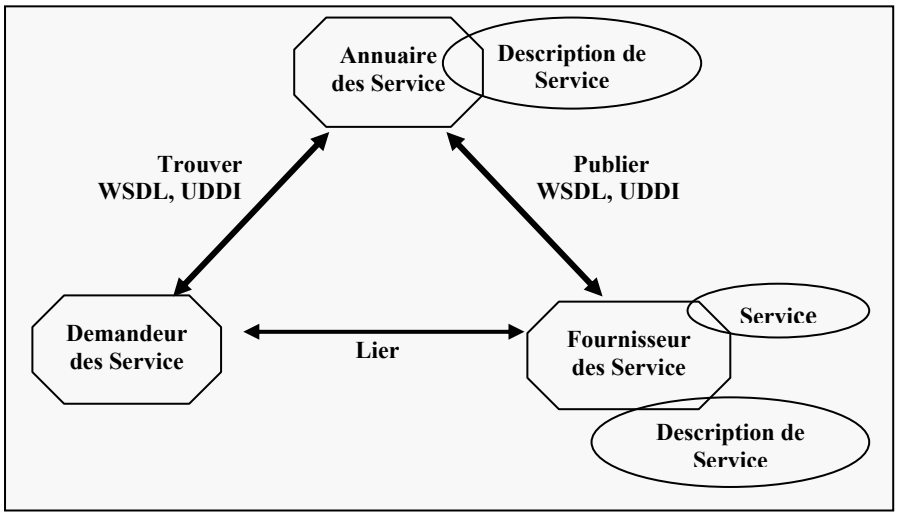

FIG. 3.8 – Invocation de services Webs.

4.3 La pile d'un Service Web

Les différentes interactions dans un service Web sont régis par les spécifications XML suivantes **:** ***[KRE 01]***

- *SOAP :* pour le transport des données et l'infrastructure de communication ;
- *WSDL :* pour la description des services offerts ;
- *UDDI :* annuaire pour le référencement des services par les fournisseurs ,et leur découverte par les utilisateurs.

Afin d'englober les différents mécanismes et notions liés au services web, *une pile architecturale* pour les services web a été proposée (figure 3.9): ***[KRE 01]***

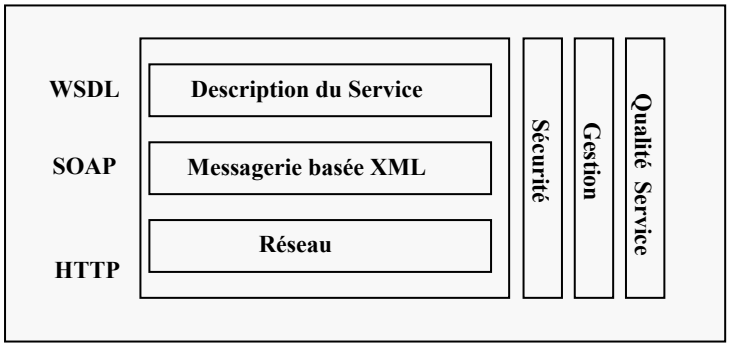

FIG. 3.9 - La pile Conceptuelle d'un service web

- La couche de base de la pile est le réseau, les Services Web doivent être accessibles via le réseau, pour qu'ils puissent être invoqués par le *Demandeurs de Service.* Grâce à sa popularité , HTTP est le protocole réseau standard utilisé par les services web à travers l'Internet. D'autres protocoles peuvent être supportés tels que SMTP, FTP…etc.

- La couche de messagerie, représente l'infrastructure de communication basée sur XML. SOAP étant le protocole basé sur XML choisi pour assurer cette tache.

- La couche de description de Service est basée sur WSDL, qui est un standard de description de services web. Il définit l'interface ainsi que le mécanisme d'interaction du service. D'autres descriptions peuvent être ajoutées en utilisant le standard UDDI.

Dans la suite de cette partie, nous détaillerons ces trois spécifications, à savoir SOAP,WSDL et UDDI.

4.4 Protocole de messagerie SOAP

4.4.1 Définition

SOAP (*Simple Object Access Protocol)* est un protocole d'échange de messages, dans un environnement distribué basé sur XML. Il utilise le protocole HTTP (ou SMTP) comme infrastructure de communication, pour envoyer et recevoir des messages dont le corps est un fichier XML. HTTP permet à SOAP , contrairement aux technologies telles que CORBA ou DCOM ,de passer à travers les Firwalls ; ce qui fait de lui le protocole le plus adapté aux applications distribuées sur Internet. SOAP consiste en trois parties : ***[GIR 01]***

- Une enveloppe qui définit un environnement pour la description du contenu, le destinataire et la nature du message.
- Un ensemble de règles qui définissent le mécanisme de sérialisation utilisé pour échanger des objets.
- Une convention pour la représentation d'un appel et réponse de procédure à distance (RPC).

4.4.2 Scénario d'utilisation de SOAP

Une interaction entre un *fournisseur de Services* et un *demandeur de services,* selon le protocole SOAP, s'effectue à travers les étapes suivantes ***[KRE 01]*:**

1. Dans la figure 3.10 (1), une application *demandeur de services* crée un message SOAP. C'est une requête qui invoque une opération Service Web, fournie par le *Fournisseur de Services.* L'environnement SOAP interagît avec le protocole sous jacent (HTTP, …etc.) pour envoyer le message SOAP à travers le réseau.

2. En (2), le réseau délivre le message au module SOAP du *fournisseur de Services.* Le serveur SOAP route le message vers le *Fournisseur* , il doit entre autre le convertir à partir du format XML vers le langage de programmation spécifique à l'application.

3. Le service Web traite la requête contenue dans le message et formule une réponse en un message SOAP, à (3) la réponse est renvoyée par le serveur SOAP.

4. En (4) , le message réponse est reçu sur le nœud du *Demandeur de services*, il est converti à partir du format XML vers le langage de programmation cible par le module SOAP.

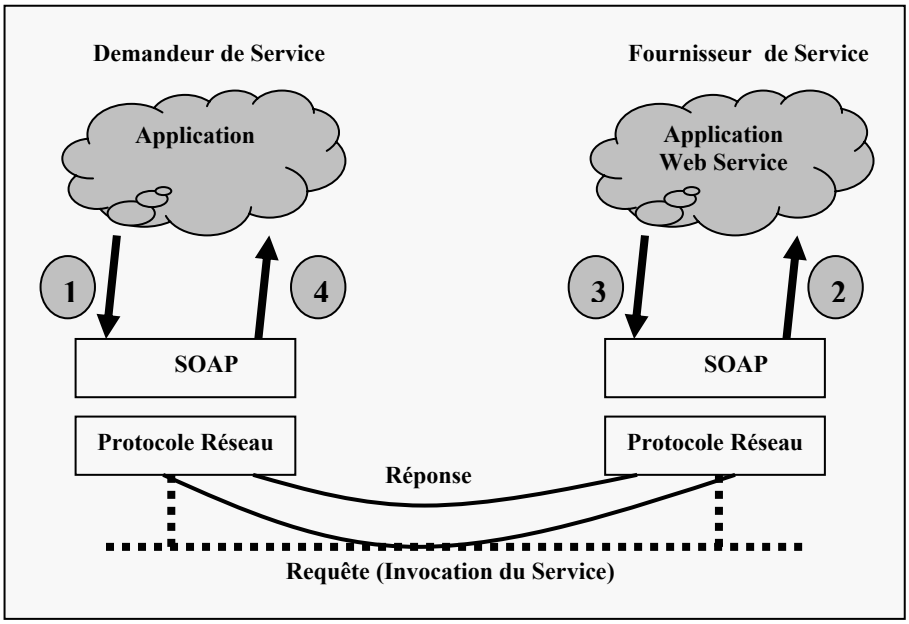

FIG. 3.10 - Messages XML en utilisant SOAP

4.5 Description des Services WSDL (Web Service Description Language)

WSDL *(Web Services Description Languages)* est pour les services Web ce qu'est l'IDL (Interface Definition Language) pour la programmation distribuée (CORBA, DCOM). Il décrit de façon précise les services Web, en incluant des détails tels que les protocoles, les serveurs, les ports utilisés, les opérations pouvant être effectuées, les formats des messages d'entrée et de sortie, et les exceptions pouvant être renvoyées. ***[GIR 01]***

C'est à travers *la description du Service* que le *Fournisseur de Service* communique - au *Demandeur de Service* -les spécifications pour l'invocation du Service Web. Un document WSDL est un document XML qui contient deux parties : *l'interface du Service* et *l'implémentation du service* (figure 3.11) ***[KRE 01]*** Tels que :

- **Type :** représente les types des paramètres des opérations.
- **Message :** représente les messages d'appel et retour de chaque opération.
- **Port type :** Groupe d'opération.
- **Binding :** URL de l'opération et Type de protocole.

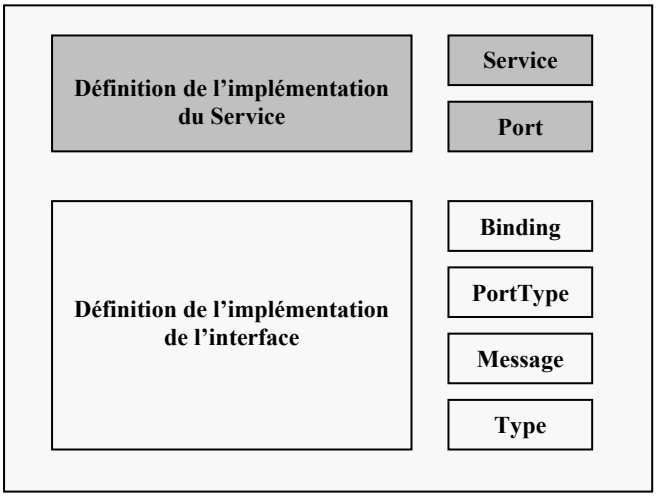

FIG. 3.11 – Les couches d'un Service Web

4.6 Annuaire des Services UDDI

4.6.1 Définition

Créé par IBM, Microsoft et Ariba ; UDDI *(Universal Description, Discovery and Integration)* est un annuaire distribué de services Web et d'entreprises (*Business/Service Registry)*. Il fournit un moyen de normaliser les échanges *fournisseurs – demandeurs* de service ; en permettant aux fournisseurs des services Web (*Business providers)*, d'enregistrer leurs services, et aux applications de rechercher les services correspondant à leurs besoins. ***[GIR 01]***

UDDI est composé de deux parties : ***[GIR 01]***

- Les interfaces d'accès à l'annuaire, et les modèles de données.
- L'UDDI Business Registry : annuaire d'entreprises et de services Web qui contient des informations au format XML, de types suivants :

  a) Pages blanches : contiennent les noms, adresses, contacts, identifiants,… des entreprises enregistrées, ainsi que des informations de catégorisation permettant de faire des recherches spécifiques, dépendant du métier de l'entreprise.

  b) Pages jaunes : contiennent les détails sur le métier de l'entreprise, les services qu'elle propose.

  c) Pages vertes : contiennent les informations techniques sur les services proposés ; telles que les références vers les spécifications des services Web, les interfaces implémentées, les contacts pour un processus particulier, description du processus en plusieurs langages, catégorisation des processus..*).*

4.6.2 Scénario d'utilisation d'UDDI

La Figure 3.12 représente un scénario classique d'utilisation de UDDI : ***[GIR 01]***

1. Les fournisseurs de services Web s'enregistrent auprès de l'UDDI *Business Registry* (sur les pages blanches et jaunes) ; et ajoutent leurs services (en complétant les pages jaunes ,et en renseignant les détails techniques de leurs services dans les pages vertes).

2. Un utilisateur recherche une entreprise fournissant un service donné, et obtient une description de l'entreprise par le biais des pages blanches et jaunes, et des détails de l'invocation du service offert par le biais des pages vertes.

3. L'utilisateur peut alors invoquer le service Web distant en utilisant SOAP.

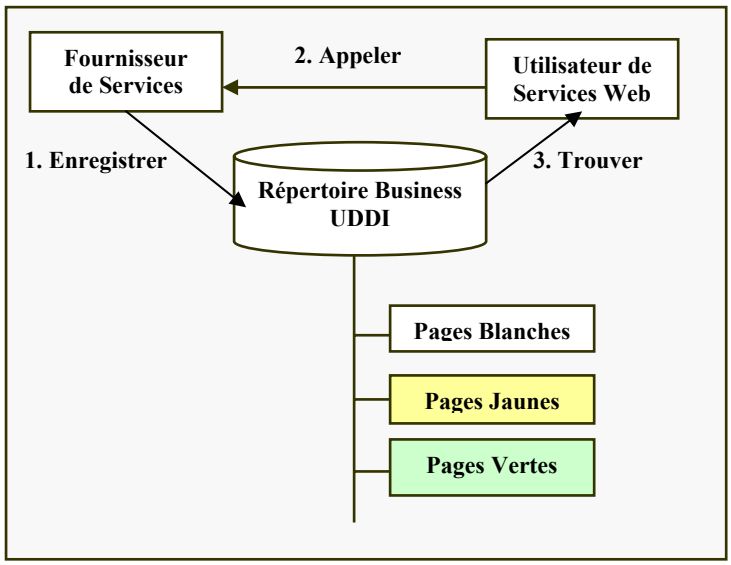

FIG. 3 .12 -Scénario Classique d'utilisation d'UDDI

4.6.3 Types d'annuaires UDDI

On peut différencier les types suivants : *[GIR 01]*

- *L'opérateur UDDI :* est un annuaire global distribué sur Internet. Il est accessible par tous. Parmi les opérateurs UDDI on compte IBM, Microsoft et HP ;
- *L'UDDI e-marketplace* : il regroupe des informations spécifiques aux entreprises d'un même corps de métier, elles s'inscrivent et enregistrent leurs services.
- *L'UDDI portail :* il se situe au niveau du firewall d'une entreprise, il est accessible de l'extérieur. C'est un moyen pour l'entreprise de publier ses services, à destination de ses partenaires, tout en gardant un contrôle total des informations publiées.
- *L'UDDI catalogue partenaire (ou UDDI B2B) :* cet annuaire UDDI est interne à une entreprise, et n'est accessible que par les applications de celle ci. On y ajoute toutes les informations contenant ses partenaires et les services qu'ils proposent.

- *L'UDDI EAI* : il est lui aussi ,interne à une entreprise, mais les informations contenues servent avant tout à faire de l'intégration d'application en interne.

- *L'UDDI de test* **:** Il est utile lors des phases de tests précédant le déploiement des services.

5. L'intégration par «entrepôt de données» : le DataWareHouse

Le data WareHousing est une alliance des différentes technologies des réseaux et des systèmes de gestion de bases de données. Il a pour but d'intégrer les données des systèmes opérationnels dans un environnement unique, en permettant l'utilisation stratégique des données. Pour ce faire , le système fournit le moyen d'extraire , à partir de l'ensemble des données de l'entreprise, des informations de prévisions et de suivi d'opérations***[RYS 02]***. Ces données sont ensuite adaptées à une classe de décideurs et stockées dans *des magasins de données.****[CAU 01].*** La figure 3.13 décrit l'architecture générale d'un entrepôt de données.***[CAU 01] [RYS 02]***

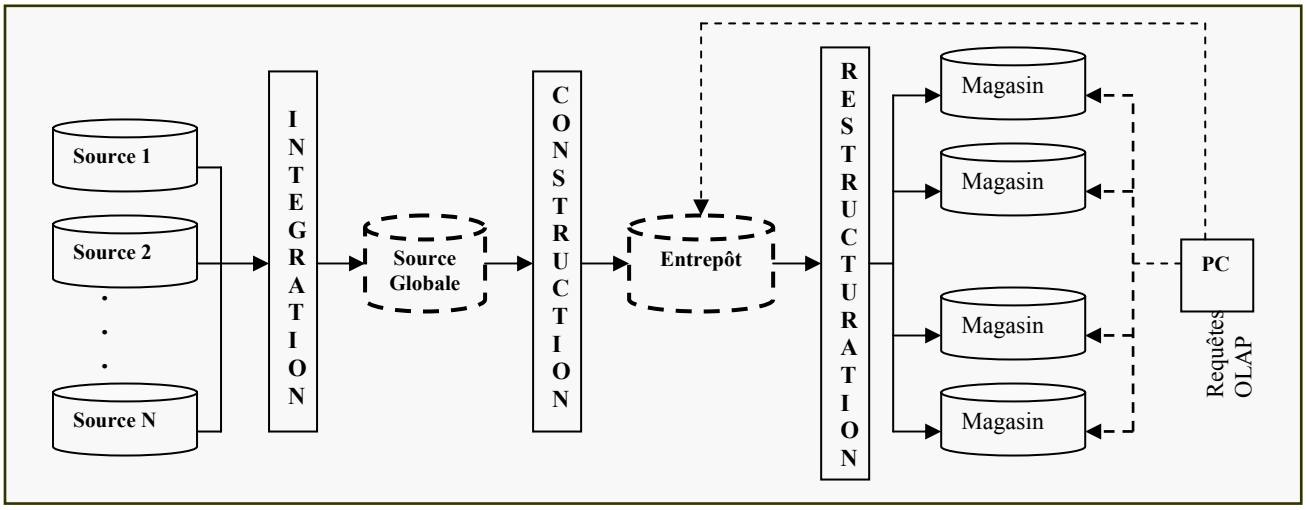

FIG. 3.13- Architecture générale d'un Entrepôt de Données

5.1 Définition

L'entrepôt de données est une collection de données orientées sujet, intégrées, non volatiles et historisées, organisées pour le support d'un processus d'aide à la décision **:** ***[CAU 01] [CN 98]***

- *Orienté sujet :* l'entrepôt de données est organisé autour des sujets majeurs de l'entreprise, où les données sont structurées par thème, pour pouvoir disposer de l'ensemble des informations utiles sur un sujet.

- *Données intégrées :* Les données qui alimentent l'entrepôt, sont issues de sources variées, pourtant elles doivent être intégrées : par exemple le même indicateur de chiffre d'affaires intéressera autant les forces de vente que le département financier ou les acheteurs.

- *Données historisées :* Contrairement aux données de productions , les données dans un entrepôt de données sont historisées ; et jamais mises à jour. Un référentiel de temps est associé à chaque donnée ; afin de pouvoir identifier une valeur particulière dans le temps.

- *Données non volatiles :* La non volatilité des données ,est une conséquence de l'historisation. Une même requête effectuée à quelques mois d'intervalle en précisant la date de référence de l'information recherchée donnera le même résultat.

5. 2 Les Composants de base d'un entrepôt de données

Un entrepôt de données comporte essentiellement les éléments suivants (figure 3.14) ***[CN 98] [RYS 02]*:**

- *Le système source* : Système opérationnel d'enregistrement. Il s'agit souvent de ce que l'on appelle les applications de gestion de l'entreprise.

- *La zone de préparation des données :*Elle regroupe l'ensemble des processus qui nettoient, transforment, combinent, archivent, suppriment les doublons. Elle prépare les données sources, en vue de leur intégration et de leur exploitation dans l'entrepôt de données.

- *Le data mart :*Le data mart est défini comme un sous-ensemble logique d'un entrepôt de données. Il représente un projet réalisable. En effet, c'est une réduction de l'entrepôt de données à un seul processus, ou à un groupe de processus ciblant un groupe métier spécifique.

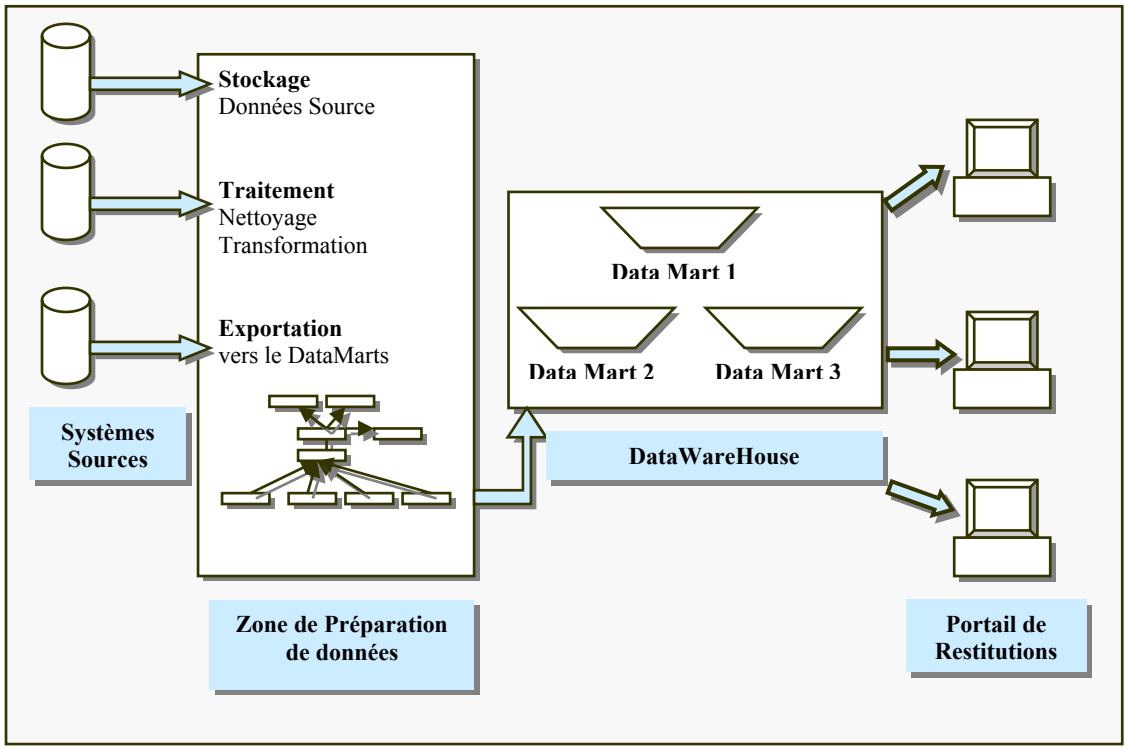

FIG. 3.14 - Les Composants d'Un entrepôt de données

- *L'entrepôt de données ou data warehouse :* l'entrepôt de données correspond à la source de données interrogeable de l'entreprise. Selon la définition du *data mart*, on peut voir l'entrepôt de données comme l'union de tous les *data mart*s qui le composent.

- *Portail de restitution :* Sous ce terme ,sont regroupées toutes les applications qui s'appuient sur les données de l'entrepôt , pour les restituer soit à l'utilisateur, soit à une autre application. Les services offerts par le portail de restitution ,sont les services d'accès aux données, les applications de modélisation et de data mining.

5.3 Modélisation des données dans un entrepôt de données

Pour modéliser un entrepôt de données, on utilise soit la modélisation classique en relationnel ; soit la modélisation dimensionnelle. ***[CAU 01] [TES 00]***

a) Modélisation relationnelle :

- *Modélisation relationnelle normalisée :* C'est le modèle classique, on y trouve beaucoup de données de détails, mais aucune information intéressante dans un contexte décisionnel n'existe directement.

- *Modélisation relationnelle dénormalisée :* dans ce cas , le modèle normalisé est simplifié, afin de réponde aux exigences du décisionnel. En créant des redondances d'informations et des informations agrégées, qui diminueront le nombre de tables, dans le but d'améliorer les temps de réponse , et de faciliter l'accès aux informations.

b) Modélisation dimensionnelle :

Dans cette approche, Les données de l'entrepôt sont modélisées selon plusieurs axes d'analyses (ou dimensions) ; pouvant représenter des notions variées, tels que le temps, la localisation géographique, le code identifiant des produits…etc. Ce format multidimensionnel est connu sous le nom d'hypercube***[CAU 01]***. Dans un hypercube, on définit la notion de faits ; qui est l'intersection d'un membre de chaque dimension ***[CAU 01]***; si on prend l'exemple d'une activité de vente, ou on modélise les dimensions, un fait de vente sera un ensemble de mesure comme la quantité vendue et le montant, pour un article donné.

Pour modéliser de telles données, deux approche existent : ***[CAU 01] [TES 00]***

- *Modélisation en étoile :* Dans un modèle en étoile, tous les faits sont définis dans une simple table relationnelle. Cette table est reliée par clés primaires à d'autres tables correspondant aux dimensions. Le modèle en étoile superpose une structure multidimensionnelle au dessus d'un modèle relationnel normalisé à deux dimensions.

- *Modélisation en Flocon :* C'est une modélisation en étoile pour laquelle, on éclate les tables de dimensions en sous-tables selon la hiérarchie de cette dimension.

5.4 Analyse multidimensionnelle : OLAP.

Les outils OLAP *(Online Analytical Processing)* sont des outils d'analyse des données stockées dans un entrepôt de données. Ils reposent sur le principe suivant: restructurer et stocker dans un format multidimensionnel les données issues de fichiers plats ou de bases relationnelles ,pour les exploiter rapidement. Parmi les règles que doivent respecter les outils OLAP : ***[TES 00][CN 98]***

- L'utilisateur ne doit pas être concerné par l'intégration des données.
- Les outils OLAP doivent avoir leur propre schéma logique de stockage des données physiques hétérogènes, doivent accéder aux données et réaliser n'importe quelle conversion afin de présenter à l'utilisateur une vue simple et cohérente. Ils doivent aussi savoir de quel type de systèmes proviennent les données.

5.5 Structure d'un entrepôt de données

Un entrepôt de données est structuré en quatre classes de données, organisées selon un axe historique et un axe synthétique (Figure 3.15).***[CN 98]***

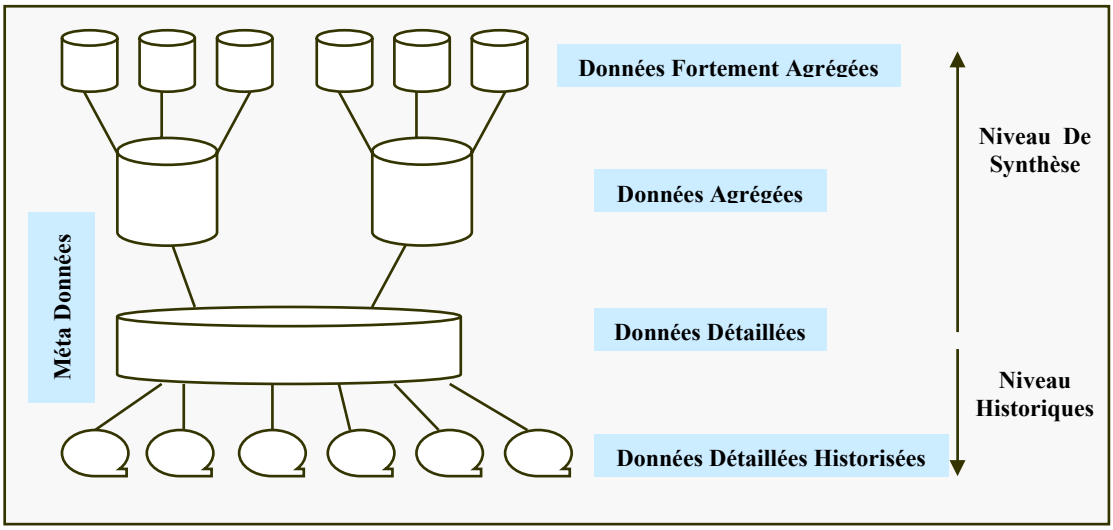

FIG. 3.15 - Différentes classes de données d'un entrepôt de données

- *Données détaillées :* Elles reflètent les événements les plus récents. Les intégrations régulières des données issues des systèmes de production vont habituellement être réalisées à ce niveau.

- *Données agrégées :*Elles correspondent à des éléments (résultats) d'analyse représentatifs des besoins utilisateurs.

- *Les méta-données :* Constituent une véritable aide en ligne permettant de connaître l'information contenue dans l'entrepôt. Les méta-données sont intégrées dans un référentiel et contiennent des informations sur la sémantique des données, leurs localisation, ainsi que des règles de description, et finalement la structure de la base qui implémente l'entrepôt de données.

- *Données historisées :* Chaque nouvelle insertion de données provenant du système de production ne détruit pas les anciennes valeurs, mais crée une nouvelle occurrence de la donnée.

5.6 Architecture d'un entrepôt de données

Dans une architecture DataWarehouse , il existe une nouvelle couche entre le client et la source de données (les serveurs) , appelée l'entrepôt de données (figure 3.16). Cet entrepôt stocke les données issues des différentes sources. Une telle architecture sert à l'intégration des données, ainsi que pour la gestion de données décisionnelles. ***[ABI 00]***

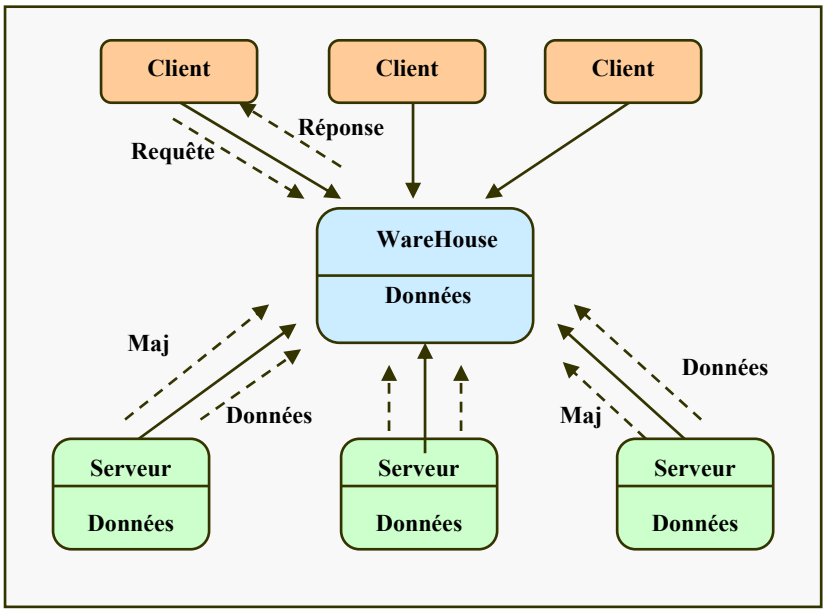

FIG. 3.16 - Architecture Générale d'un Entrepôt de données.

Pour implémenter un entrepôt de données, trois types d'architectures sont possibles **:*[CN 98]***

- *Architecture réelle :* Elle est généralement retenue pour les systèmes décisionnels. Le stockage des données est réalisé dans un SGBD séparé du système de production. Le SGBD est alimenté par des extractions périodiques. Avant le chargement, les données subissent d'importants processus d'intégration, de nettoyage, de transformation. L'avantage est de disposer de données préparées, pour les besoins de la décision et répondant aux objectifs du Data Warehouse. Les inconvénients sont le coût de stockage supplémentaire et le manque d'accès en temps réel.

- *Architecture virtuelle :* Cette architecture n'est pratiquement pas utilisée pour le Data Warehouse. Les données résident dans le système de production. Elles sont rendues visibles par des produits *MiddleWare* ou par des passerelles. Il en résulte deux avantages : pas de coût de stockage supplémentaire et l'accès se fait en temps réel. L'inconvénient est que les données ne sont pas préparées.

- *Architecture Remote :* C'est une combinaison de l'architecture réelle et de l'architecture virtuelle. Elle est rarement utilisée. L'objectif est d'implémenter physiquement les

niveaux agrégés afin d'en faciliter l'accès, et de garder le niveau de détail dans le système de production en y donnant l'accès par le biais de *MiddleWare* ou de passerelle.

5.7 L'Alimentation d'un entrepôt de données

Alimenter un entrepôt de données, est la difficulté technique majeure et la plus coûteuse. En effet, les données sont pour la plupart, issues de sources de données différentes. Des solutions logicielles sont alors nécessaires à leur intégration et à leur homogénéisation. Celles-ci peuvent aller de l'écriture de *batch*, à l'utilisation de logiciels spécialisés dans l'extraction et la transformation d'informations (ETI, Prism, Carleton, ...). Ces outils ont pour objectif de s'assurer de la cohérence des données de l'entrepôt de données ;et d'homogénéiser les différents formats trouvés dans les bases de données opérationnelles.

Les phases d'alimentation d'un entrepôt de données sont : ***[CN 98]***

(1) → Découvrir quelles sont les données à faire migrer.
(2) → L'acquisition des données, qui se déroule en trois phases : l'extraction, la transformation et le chargement

5.7.1 La découverte des données

La découverte des données consiste à les localiser dans le système opérationnel et à prendre les plus judicieuses. Un mauvais choix des données a extraire, risque de compliquer la phase de l'alimentation

5.7.2 l'acquisition des données ETL

Elle se compose de trois phases : l'extraction, la préparation et le chargement (figure 3.17). ***[CAU 01] [GAR 03a]***

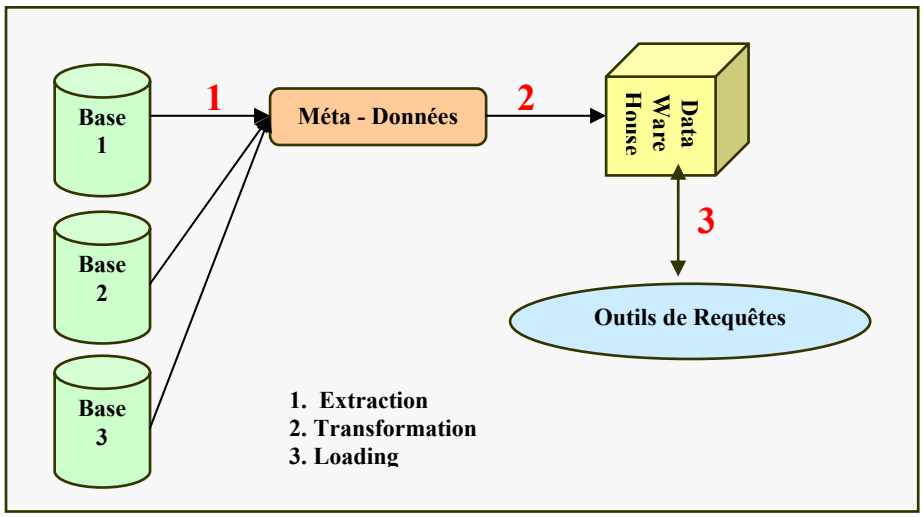

FIG. 3.17 – Les trois phases de l'ETL

A) l'extraction des données

Elle consiste à collecter les données utiles dans le système de production, afin de rafraîchir la base décisionnelle. Il faut identifier les données ayant évolué ,afin d'extraire le minimum de données, en les marquant (date dans un fichier LOG). Dans un environnement très hétérogènes les problèmes suivants apparaissent : ***[CN 98] [TES 00]***

- Les différents processus d'extraction doivent être synchronisés pour assurer l'intégrité des données.

- Lors du traitement des données externes. Il faut maintenir une surveillance du système d'information pour pouvoir identifier des données et s'assurer que ce sont bien celles qui sont recensées. De plus, ces données nécessitent un reformatage pour les incorporer dans une forme exploitable pour l'entreprise.

- L'outil d'extraction doit être capable de traiter toutes sortes de sources de données sans être perturbé et s'adapter aux différentes évolutions du système.

Pour extraire les données sources, on utilise les technologies suivantes : ***[TES 00]***

- Des passerelles, fournies principalement par les éditeurs de bases de données. Ces passerelles sont généralement insuffisantes, car elles sont mal adaptées aux processus de transformation complexes.

- Des utilitaires de réplication, utilisables si les systèmes de production et décisionnel, sont homogènes et si la transformation à appliquer aux données est légère.

- Des outils spécifiques d'extraction. Ces outils représentent l'inconvénient de leur prix relativement élevé.

B) La transformation des données

La transformation ou Le nettoyage des données consiste d'auditer et d'éventuellement améliorer la qualité des données. En effectuant les taches suivantes : ***[CN 98][GAR 03a]***

- Supprimer les doublons dans les fichiers**.**
- Appliquer des filtres prédéfinis sur les données, afin d'attribuer des valeurs cohérentes aux variables mal ou non renseignées.
- Harmoniser les formats (date : jj/mm/aaaa).
- Pouvoir associer des champs sources avec des champs cibles. Exemple :

    a) Le transfert du "*nom du client*" vers un champ cible.
    b) La décomposition d'une "*adresse*" vers les champs "*numéro", "rue", "ville"* ou l'inverse.

- Réaliser des analyses lexicales des champs sources. Pour comprendre, par exemple que les champs suivants signifient la même chose *: "Boulvd", "Bd" ,"Boulevard"*.

- En complément, on trouve des outils d'audit et d'analyse pour assurer le suivi du processus afin notamment de contrôler les rejets.

C) Le chargement des données

Le chargement est la dernière phase de l'alimentation d'un entrepôt de données. C'est une phase délicate notamment lorsque le volume des données est important. Pour obtenir de bonnes performances en chargement, il est impératif de : ***[CN 98]***

- Maîtriser les structures du SGBD (tables et index) ,associées aux données chargées afin d'optimiser au mieux ces processus.

- Optimiser les chargements lourds des données, en procédant de manière parallèle , en utilisant des utilitaires particuliers .

La synchronisation des chargements, de la source vers la cible est un problème majeur. Pour réaliser ce transfert des données, on utilise deux techniques : ***[CAU 01]***

- *Le transfert de fichiers :* Qui consiste à créer au moment de l'extraction des données, un ensemble de fichiers qui seront transférés sur le système cible, afin d'alimenter la base décisionnelle. La transformation des données s'effectuera alors soit à la constitution des fichiers, soit à leur arrivée sur la cible.

- *Le transfert de base à base :* Cette technique est plus complexe puisqu'elle implique que chaque donnée soit extraite de la base source, nettoyée si nécessaire, puis transférée sur la base cible.

Des logiciels contrôlent souvent les phases d'extraction, de transport et de chargement. Ils créent un enchaînement automatisé et gèrent les mises à jour, mais ils sont rarement capables de s'occuper également du nettoyage des données. Il faut donc souvent combiner les outils pour alimenter un DataWarehouse***[CN 98]***. Le tableau suivant (figure 3.18) illustre les principaux outils d'extraction et de nettoyage qui existent sur le marché : ***[CN 98]***

Editeur	Produit
EXTRACT SUIT	ETI
PASSPORT	CARLETON
WAREHOUSE MANAGER	PRISM SOLUTION
VALITY TECHNOLOGIE	INTEGRITY DATA REENGINRRRING

FIG. 3.18 - Principaux Outils d'extraction et de nettoyage

6. Synthèse : Les bases de données fédérées.

Dans une approche de bases de données fédérées, les données intégrées ne sont pas matérialisées dans un SGBD. L'intégration se fait sur demande, pour répondre aux requêtes » ***[GAR 99b].*** **G.Gardarin** définit *les bases de données fédérées* ***[GAR 99b]*** comme suit : « Une base de données fédérée est une base de données distribuée hétérogène constituée à partir de sources de données de natures variées : fichiers classiques, textes , document HTML, XML, BD relationnelle ou objet. Le but étant de fournir une vue intégrée de différentes données de l'entreprise » (figure 3.19) .

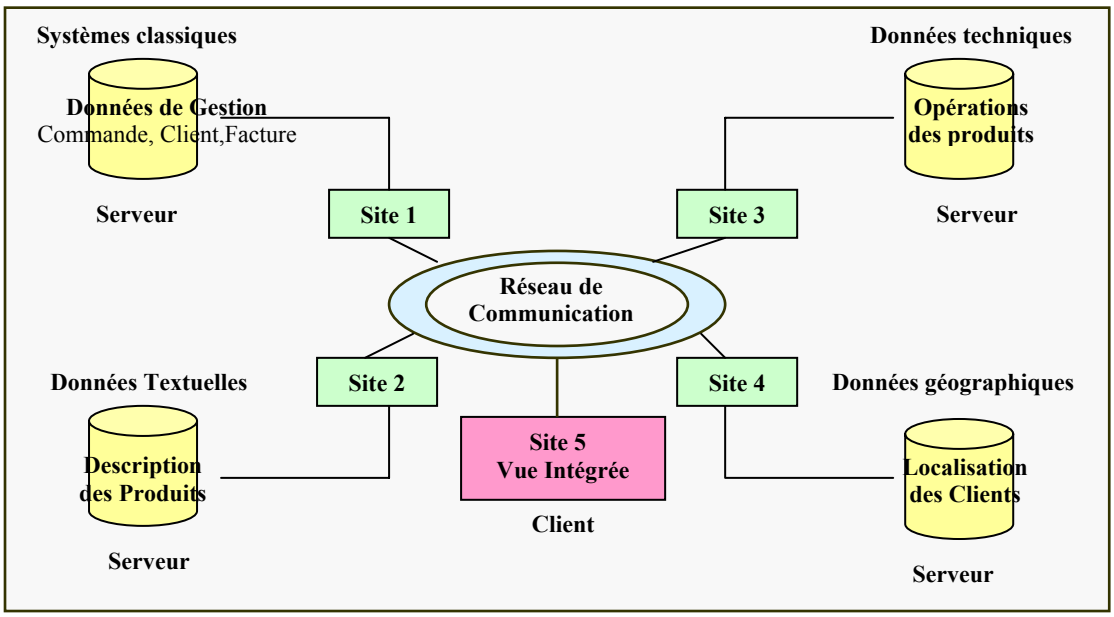

FIG. 3.19 - Exemple de BD fédérées .

Dans cette catégorie, on distingue les outils de *l'Entreprise Information Integration* (EII). Il s'agit d'un environnement d'intégration spécialisé dans l'accès aux données. Les outils EII regroupent au sein de vues métier homogènes, des données provenant de nombreuses sources hétérogènes (bases de données relationnelles, et non relationnelles, fichiers, XML, etc.). ***[XF 01]***

L'intégration via les outils EAI se distingue alors, d'un système de gestion de bases de données fédérées, avec son caractère asynchrone ,et surtout parce qu'elle est basée sur l'harmonisation d'API (i.e. de services applicatifs), et non sur celle des modèles de données. Elle est à privilégier dans les cas de grande diversité technique des sources (données et/ou traitements), avec le plus souvent, des contraintes transactionnelles et volumétriques fortes. Elle nécessite néanmoins des investissements lourds.

La solution « Services Webs » se distingue aussi des bases de données fédérées ; puisque l'objectif est surtout l'harmonisation d'appel de services (et non celle des données) sur un protocole et un vocabulaire unique. Elle reste à privilégier pour des besoins d'intégration légère (verbosité textuelle, fréquence d'échange limitée, appel d'un traitement distant en boite noire). Néanmoins, les services webs peuvent être utilisés pour implanter des solutions de systèmes de fédérations de données.

Enfin, Les outils ETL constituent un excellent mécanisme pour intégrer des données hétérogènes. Cependant, leurs objectifs est de créer une nouvelle base de données, pas de diriger des requêtes vers différents systèmes hétérogènes. L'inconvénient principal des entrepôts de données par rapport à la technique temps réel des bases de données fédérées, devient la pertinence (la fraîcheur) des informations répliquées. Néanmoins, ce principe est aujourd'hui utilisé dans la majorité des applications à caractère décisionnel, du fait notamment de son faible impact en performance sur les données de production.

7. Conclusion

Dans ce chapitre, on a vu les différentes techniques d'intégration d'information au niveau des systèmes informatiques. Ces techniques donnent un nouveau souffle aux bases de données distribuées. En effet, l'EAI résout les problèmes d'intégration des applications d'une entreprise ; il est basé sur des mécanismes de messagerie puissants, capables de faire communiquer plusieurs applications hétérogènes. Les services web, quant à aux ,permettent d'exporter les services d'une entreprises en utilisant des composants , qu'on peut invoquer à distance. Enfin, les entrepôt de données représentent une véritable alternative au bases de données distribuées ; puisqu'ils permettent d'intégrer diverses sources hétérogènes dans une base unique. Cependant , la fraîcheur des données restent un point critique non résolu par cette approche.

Chacune des approches citées dans ce chapitre présente des solutions partielles aux problèmes d'intégration de données issues de sources hétérogènes et disparates. Par conséquent, il est impératif dans la majorité des cas, de combiner ces trois approches ; ce qui peut s'avérer lourd et coûteux pour une entreprise.

Chapitre 4 : XML Comme Modèle de données

1. Introduction

XML (*eXtensible Markup Language*) est le nouveau standard adopté par Le World Wide Web Constorium (W3C) afin de compléter HTML pour l'échange des données sur le Web.

En effet, HTML à été conçu à l'origine pour la présentation des données sur le Web, ses balises prédéfinies ne permettent pas une manipulation aisée du contenu des pages HTML. De ce fait , le langage XML a été défini pour permettre de décrire les données. La description est alors incluse avec les données d'où le terme de données *auto-descriptives.* La description des données à permis, de rajouter une certaine structure aux données. On est loin du cas des données structurées en relationnel ou objet, mais plutôt de données *semi- structurées.*

Les données semi structurées ont récemment émergé comme un thème important dans un contexte de données hétérogènes tel que le Web. Dans ce cas, où l'information - qui est normalement associée avec un schéma précis - est contenue à l'intérieur des données. En contrepartie, de telles données sont difficiles à manipuler par des traitements automatiques. Des modèles de données propres aux données semi structurées ont été alors proposés.

Le but de ce chapitre est de présenter le langage XML du point de vue des bases de données. Ainsi nous décririons les principaux concepts qui lui ont permis, de jouer un rôle primordial dans les nouvelles architectures des bases de données fédérées.

Nous introduisons brièvement les différents aspects de ce langage dans la section (2) . Dans la section (3) , on décrit la syntaxe XML . Dans la section (4) , on décrit comment XML modélise les données semi structurées. Les différentes techniques de description de méta données pour XML sont décrites dans la section (5). La section (6) présente les interfaces de programmation pour la manipulation des documents XML. On conclut en (7).

2. Les données semi structurées

Quand on stocke des données, on commence par décrire leurs structures (Schémas). Ensuite , on crée des instances de ces structures en remplissant les schémas ***[ABI 00].*** Cependant , stockées de manière fortement structurée, ces données ne permettent pas , dans tous les cas , de déduire le monde réel. Car la majorité des données du monde réel n'ont pas de structures régulières et statiques, comme cela est le cas pour les données relationnelles ou objets. Ce qui peut entraîner les problèmes suivants : ***[BOU 02b] [DAN 03a]***

- *La structure de données évolue :* cela nécessite alors la modification du schéma des données.
- *Les données ne sont pas forcément conformes au schéma :* ce qui implique l'utilisation de beaucoup de valeurs nulles.
- *Le même attribut peut avoir des types différents suivant les données :* ce qui oblige l'utilisation de type englobant (chaîne de caractères) , qui entraîne une réduction dans la finesse de la description.
- *On peut avoir une instance d'attribut multi- valuée ou mono-valuée.*
- *Les données peuvent être faiblement structurées :* texte.

Donc, de telles données, caractérisées par une structure irrégulière, dynamique ou inconnue (telles les données en provenances du web) sont dites *données semi structurées.* Ils sont souvent appelées aussi données *sans schéma* ou *auto descripteur*. En d'autres termes, la description des données n'est pas séparée de leur structure de stockage.

2.1 Motivations des données semi structurées

Les données semi structurées sont une solution efficace, pour représenter les données qui ne peuvent pas avoir de modèle conventionnel (Relationnel, Objet). En d'autres termes, ils peuvent être utilisées pour résoudre les points suivants : ***[BUN 97]***

- *Des données non structurées :* un exemple de ce type de donnée sont les documents. En effet, les documents contiennent du texte, et les langages de requêtes traditionnels (SQL/OQL) ne sont pas suffisamment expressifs pour atteindre les informations incluses dedans. Un deuxième exemple, le plus important , est celui du Web. En effet, les données en provenance du web ne peuvent pas être soumises à un schéma. La plupart des requêtes sur le web ne font que retirer des pages individuelles.

- *Manipulation des méta-données :* alors que les langages de requêtes traditionnels (SQL/OQL) ne permettent pas de manipuler aisément les méta données d'une base de données ; les données semi structurées le permettent facilement. C'est une fonctionnalité très utile aux programmeurs pour comprendre le schéma d'une base de données, est de fournir un moyen de naviguer dans le schéma. La figure 4.1 , illustre les types d'informations qu'on peut obtenir.

Q1 : Existent t-ils dans la base de données des entiers > 2^{16} ?

Q2 : Quels sont les objets dans la base de données qui ont le nom d'attribut commençant par « Act » ?

FIG. 4.1 Manipulation des méta données

- *Intégration des données :* Intégrer plusieurs données hétérogènes , nécessitent l'utilisation d'un format d'échange et de transformation commun. Dans ce contexte , le modèle *des données semi structurées* ***[ABI 00]*** offre un environnement capable de prendre en charge des types de données différents. En effet, plusieurs modèles (relationnel, objet, fichier plat, réseau, hiérarchique) peuvent être facilement représentés, dans un même modèle de données semi structurées.

2.2 Les caractéristiques des données semi structurées

Parmi les caractéristiques ***[ABI 00] [BOU 02b] [DAN 03a]*** des données semi structurées, on peut citer les suivants :

- *La structure est irrégulière :* une même information peut être représentée par plusieurs éléments hétérogènes dans la même collection. Un même attribut peut être *mono-valué* dans une instance et *multi-valué* dans d'autres. Des éléments peuvent manquer dans certains endroits.

- *La structure est implicite :* les données semi structurées ne permettent pas de déduire facilement leurs structures. L'extraction et l'interprétation de la structure est un processus difficile puisqu'il s'agit à la fois d'analyser , d'interpréter les données et d'effectuer des correspondances logiques pour déduire la structure.

- *La structure est partielle :* une partie des données peut être constituée d'informations non structurées (images , textes bruts).

- *Le typage est irrégulier :* et ceci est dû à l'hétérogénéité des données.

- *Le schéma est lâche :* la consistance des données dans les bases de données relationnelles est protégée en utilisant un typage stricte ; dans les données semi structurées des transgressions sont tolérées, ce qui conduit à une altération des schémas.

- *Le schéma peut être antérieur ou postérieur :* par rapport à l'existence des données.

- *Le schéma est large :* en conséquence de l'hétérogénéité des données, le schéma est généralement très large pour englober toutes les informations des instances des données.

- *Le schéma évolue rapidement :* les sources de données semi structurées sont dynamiques, ce qui fait que leur organisation change fréquemment. En conséquence , leurs schémas sont souvent mis à jour.

Ces caractéristiques permettent aux données semi structurées de modéliser une infinité de données, hétérogènes au départ, dans un format unique. D'où leur utilisation comme modèle de données commun dans les architectures de fédération de données.

2.3 La Syntaxe des données semi structurées

2.3.1 Le Syntaxe de base

Dans le modèle semi structuré, on représente trois types de données de bases : *les chaînes de caractère (String), les nombres entiers (Integer) et les Etiquettes (Labels).* Les données sont décrites directement avec la syntaxe suivante : ***[ABI 00]***

- Les listes associatives qui sont des paires de *Etiquette-Valeure*, sont utilisées pour représenter les données semi structurées (figure 4.2 , (a))
- Les valeurs peuvent être elles mêmes d'autres listes (figure 4.2, (b))
- Ajouter dans le concept des listes associatives, la possibilité d'avoir des listes avec des Etiquettes dupliquées, afin de pouvoir décrire plusieurs tuples d'une relation (figure 4.2, (c))

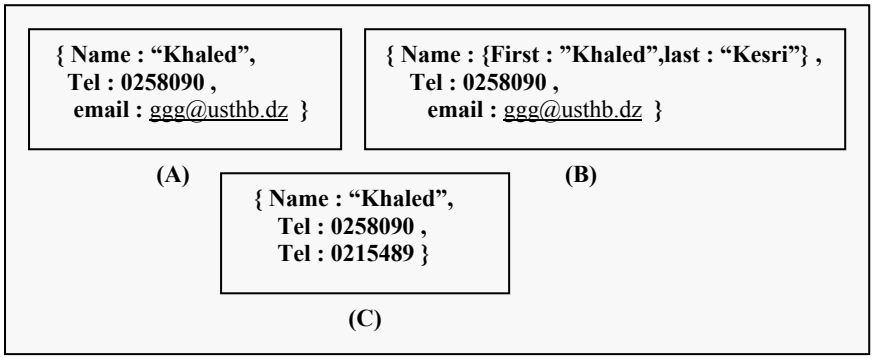

FIG. 4.2 – Syntaxe du modèle semi structuré

2.3.2 La grammaire

Les expressions des données semi structurées sont désignées par la notation suivante : *ssd-Expression,* la grammaire présentée dans la figure 4.3. décrit une syntaxe pour les types de données de base (les valeurs atomiques : chaînes de caractère et nombre, et les étiquettes**).*[ABI 00]***

< SSD-Expression > :: = < Valeur > / OID <Valeur> /OID
<Valeur > :: = Atomique_valeur / <Complex_Valeur >
<Complexe_Valeur > ::= { Etiquette : <SSD-Expression > ,…,Etiquette : <SSD-Expression > }

Ou *OID* est l'identifiant d'un objet .

FIG. 4.3 – La grammaire des données semi structurées

2.4 Modélisation des données semi structurées

L'idée de base qu'exprime les données semi structurées est la représentation des données dans une structure sous forme de *Graphe* ou *arbre.* ***[ABI 00].*** Ainsi, les données de l'exemple précédent (figure 4.2 ,(a)), sont présentées sous forme d'arbre dans la figure 4.4 suivant :

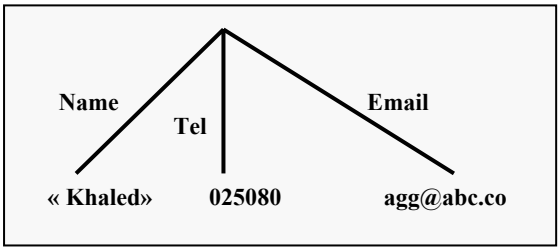

FIG. 4.4 - Graphe de données semi structurées

2.4.1 La Sérialisation des données semi structurées

Une caractéristique importante des données semi structurées, est qu'elles peuvent avoir des variations dans la structure. Le traitement des données est effectué dans ce cas, sur des graphes complètement aléatoires. Les variations de structure se résument dans *la perte de données* , *la duplication des champs , …etc.*

L'une des techniques utilisées, pour permettre la représentation de données à structure aléatoire, est de ne pas typer les données en les *sérialisant* ***[ABI 00]****;* c'est à dire en annotant chaque donnée avec sa description (comme *Nom, Phone…etc.*).D 'où l'appellation *Données auto descriptives (*Figure 4.5 *)* . ***[ABI 00]***

```
{ Personne
   { Nom : « Khaled»  , phone :0254086,  email : « test@yahoo.fr »}  ,
     Personne : {  Nom : {First : « sarah » ,  last : « toumi »} ,
                    Phone : 0273040,
                     email : « test@yahoo.fr »    ,
                    }
     Personne : { Nom : « Karim  »,
                   phone : 0254090,
                    height : 183 }
}
```

FIG. 4.5 Variation de structure des données Semi structurées

*La sérialisation des données****[ABI 00]*** est la conversion de celles ci, dans un flux de paquets (*Byte stream*) qui peut être transmis, et reconstruit au niveau du récepteur facilement. Une telle représentation peut être coûteuse en espace, puisque le description est répétée pour chaque donnée ; cependant, elle permet l'interopérabilité et le complément d'information perdue par la suppression du typage.

2.4.2 Le modèle d'échange d'objet (*The Object Exchange model)* OEM

La première utilisation des données semi structurées à été pour l'échange des données entre applications. Le modèle de format d'échange d'objet : *OEM* a été utilisé ; OEM a été défini explicitement dans le cadre du projet *TSIMMIS* ***[MOL 95] [MOL 97] ,***un système pour l'intégration de données issues de sources hétérogènes.

Dans ce modèle , la base de données a été représentée comme un graphe labellisé , où l'information du schéma est contenue dans les labels ***[MOL 95] [MOL 97]***. Un objet OEM est un quadruple *(Label, Oid, Type, Valeur)* où :

- *Label :* est de type chaîne de caractère (String).
- *Oid :* est l'identifiant de l'objet.
- Le type de données est soit complexe, et dans ce cas l'objet est appelé *Objet Complexe*, et sa valeur est un ensemble de *OIDs.* L'autre cas, est que l'objet soit de type *atomique* de ce type de donnée.

Dans la figure 4.6 , le graphe OEM a les arcs labellisés avec des données de type entier, chaîne de caractère (ou de type de base quelconque :exp. Vidéo, audio…etc.). les arcs sont aussi labellisés par des noms appelé *Symbole*, comme FILME ou TITRE .

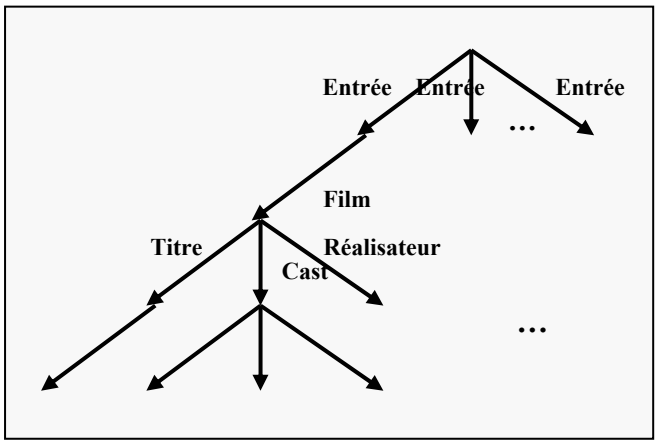

FIG. 4.6 - Modèle OEM d'une bd sur les filmes

Le modèle OEM présente les avantages suivants : ***[ABI 00]***

- Facilité d'intégration des données hétérogènes.
- Facilité de traitement des requêtes, sans avoir à connaître le type de données.
- Facilité de traduction des schémas relationnels et orientés objets dans ce modèle.

Cependant , il présente les inconvénients suivants : ***[ABI 00]***

- La perte de l'information sur le type de données.
- Difficulté d'appliquer les techniques d'optimisations.

3. XML

XML (*Extensible Markup Language*) est un sous ensemble du langage SGML *(Standard Generalized Markup Langage)* définit par l'ISO (ISO 8879). Il a été conçu spécialement, pour décrire plutôt le contenu que la présentation des documents XML. En effet, le développement du rendu de la page XML est traité par des langages spécifiques tels que *XSL (eXtended Style Sheet)* ***[STR 00].***

XML est un langage à balises, qui définit un format universel de représentation des données. Un document XML contient à la fois des données et les indications sur le rôle que jouent ces données. Ces indications permettent de déterminer la structure du document. ***[GIR 01].***

XML diffère d'un document HTML en trois aspects ***[ABI 00]*** **:**

- Des nouvelles balises peuvent être définies selon le besoin.
- Les structures peuvent être imbriquées à de différentes profondeurs.
- Un document XML peut contenir une description optionnelle de sa grammaire.

3.1 La Syntaxe de base d'un document XML

XML est une représentation textuelle des données. Le composant de base en XML est l'élément *Elément.* C'est du texte encadré par une balise *Ouvrante* comme <*Personne*> et une

balise *Fermante* </Personne>. A l'intérieur d'un *Elément*, on peut trouver des lignes de texte (*Content)*, d'autres *Eléments*, ou les deux. (figure 4.7).

```
<Personne>
   <Nom> Linda</Nom>
   <Age> 56</Age>
   <Email> Lin@usthb.dz </Email>
</Personne>
```

FIG. 4.7 – Exemple de document XML

Le type de données dans un document XML est généralement de type *PCDATA (Parsed Character Data)*. Ce type de données à été spécialement développé afin de permettre l'échange de données dans plusieurs langages ***[ABI 00].***

Un *Elément* peut contenir des informations additionnelles appelées *Attributs (Atributes)*. Un *Attribut* est un couple formé d'un nom et d'une valeur et est représenté à l'intérieur de la balise ouvrante sous la forme *nom attribut ="Valeur "* .

3.2 Structure d'un Document XML

La structure d'un document XML peut être définie par deux normes :

- *La spécification XML :* est un ensemble de règles applicables, par défaut à la création de tous les documents XML. Un document qui respecte ces règles est dit *bien formé*. Parmi ces règles on cite : ***[STR 00]***

  1. Il existe un et un seul élément racine qui contient les autres éléments.
  2. Chaque balise ouvrante, à une balise fermante associée ,et il n'y a pas de chevauchement.
  3. Les noms des balises doivent contenir au moins une lettre.
  4. Les attributs des balises, s'ils existent , doivent comporter obligatoirement une valeur, qui doit toujours être entre double apostrophes.
  5. Quand un élément est vide , les balises peuvent être simplifiées : <balise></balise> est identique à </balise>

- *Normes spécifiées dans la DTD :* DTD (*Définition de Type de Données*), cette norme est facultative. Tout document qui respecte les règles décrites dans la DTD sont dit *valides* (nom des éléments, type, répétition et ordre d'apparition dans le document).

Un document XML est divisé en deux parties : une partie appelée le *prolog* ,déclare les noms des éléments, leurs attributs, ainsi que les règles de construction pour valider le balisage des données. Et une partie appelée *l'instance de document* ,contenant les marquages eux mêmes. Donc, le marquage XML contenus dans l'instance du document est valide seulement s'il est conforme aux règles décrites dans le prolog du document. ***[EDW 97]***

3.3 Les Espaces de Noms

Les "espaces de noms" (*XML Namespaces*) sont spécifiés dans une recommandation du W3C. Ils permettent de : ***[GIR 01]***

- Mélanger du vocabulaire XML provenant de plusieurs grammaires.
- Identifier de manière unique les balises XML.

Pour illustrer l'utilisation des *espaces de noms* XML, prenons le cas d'une entreprise décrite par deux documents XML. Un document qui contient les informations générales de l'entreprise (figure 4.8, (a)), et un document qui contient les informations sur son personnel (figure 4.8, (b)) : ***[GIR 01]***

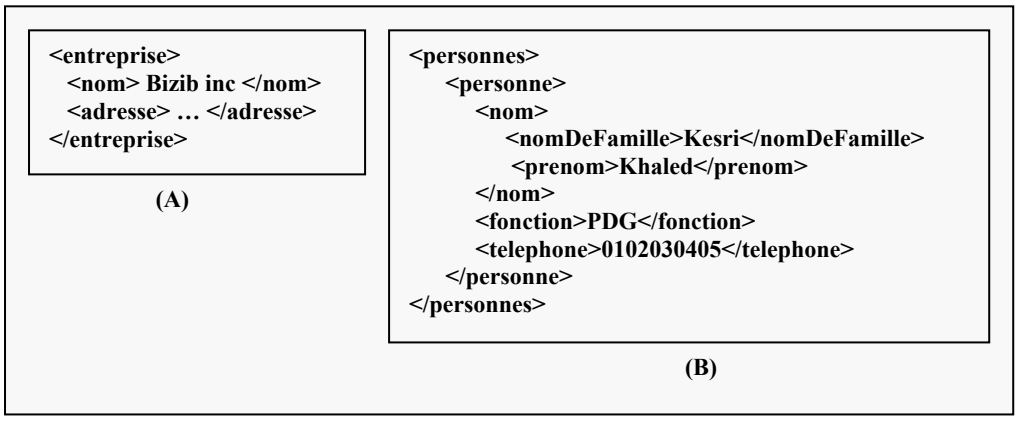

FIG. 4.8 Données XML d'une entreprise

On remarque que la balise <nom> peut contenir deux types d'informations : nom de l'entreprise ou nom d'une personne. Si l'on désire créer un document unique décrivant l'entreprise et ses employés, un moyen permettant d'identifier les balises est nécessaire . C'est l 'objectif des *XML Namespaces* (xmlns). Le principe est d'associer une *URI (Uniform Resource Identifier)* à un nom.

Par exemple l'espace de nom "personne" peut être déclaré de la façon suivante: *xmlns:personne="http://www.personne.org".* Ce nom est ensuite utilisé pour caractériser les balises. Par exemple <personne:nom> signifie que la balise <nom> appartient à l'espace de nom "personne". Le document XML unique décrivant l'organisation de l'entreprise est alors : (figure 4.9) .

```
<organisation xmlns:entreprise="http://www.entreprise.org" xmlns:personne ="http://www.personne.org">
   <entreprise:nom>Bizib inc</entreprise:nom>
   <personne:nom>
      <personne:nomDeFamille>Kesri</personne:nomDeFamille>
      <personne:prénom>Khaled</personne:prénom>
   </personne:nom>
   <personne:fonction>PDG</personne:fonction>
</organisation>
```

FIG. 4.9 - Document XML avec *espaces de noms*

3.4 Navigation et Liens

Comme HTML, XML fournit un mécanisme permettant de lier les ressources, entre autre : *XLink*, *XML Base* et *XPointer*. ***[GIR 01]***

3.4.1 XLink

XLink est une recommandation W3C qui permet de modéliser les liens entre documents. Il définit une liste d'attributs, qu'il est possible d'utiliser dés la définition de l'espace de noms XLink : ***[GIR 01]***

```
<?xml version="1.0"?>
<mondocument  xmlns:xlink="http://www.w3.org/1999/xlink/namespace"/>
```

Les attributs principaux sont :

- xlink:type : type du lien (simple, extended, arc, …).
- xlink:href : pointeur vers une ressource extérieure.
- xlink:title : titre du lien, permet de le caractériser de manière sémantique.
- xlink:role : description du lien, la forme est libre.
- xlink:from, to : utilisés dans les arcs pour définir une relation unidirectionnelle entre deux liens (du lien « from » vers le lien « to »).

Il existe trois types de liens *XLINK* : ***[GIR 01][STR 00]***

- *Les liens simples (xlink:type="simple") :* les liens simples sont une extension des liens HTTP. (figure 4.10, (a))

- *Les liens étendus (xlink:type="extended") :* ils permettent de définir des liens plus complexes (liens pointant vers des ressources multiples,…etc.). Exp. un document est disponible en plusieurs endroits ; un lien étendu peut être utilisé , couplé avec des liens de type *locator*, pour pointer sur plusieurs ressources. (figure 4.10 (b)).

- *Les arcs (xlink:type="arc") :* permettent de définir des relations entre les liens.

```
<document titre="Livre XML">
   <adresse xlink:type="simple" xlink:title="adresse " xlink:href="http://www.site.com">
              Télécharger sur le site
   </adresse>
</document>
```

(A)

```
<document titre="Livre  Java">
    <localisation xlink:type="extended">
    <adresse xlink:type="locator" xlink:role="adr princi" xlink:href="http://site.com"/>
    <adresse xlink:type="locator" xlink:role="mirroir" xlink:href="http://www.site2.com"/>
  </localisation>
</document>
```

(B)

FIG. 4.10 – Exemple de lien Xlink

3.4.2 XML Base

XML Base ajoute un attribut *xml:base* à un élément, pour définir l'adresse de base de tous les liens définis dans ses sous éléments. ***[STR 00] [GIR 01]***

3.4.3 XPointer

XPointer permet de pointer sur des éléments spécifiques d'un document XML, en utilisant des expressions de chemins XPath: ***[STR 00] [GIR 01]***. Dans la figure 4.11, le lien pointe sur l'entrée de l'annuaire « *annuaire.xml »* contenant le numéro de téléphone de *kesri khaled*

```
<link xmlns:xlink="http://www.exp.com/xlink" xlink:type="simple" xlink:href=
"annuaire.xml#xpointer(//entree[nom="Kesri khaled"])">
Numéro de téléphone de kesri khaled dans l'annuaire
</link>
```

FIG. 4.11 – Exemple de lien XPointer

4. XML comme modèle de données semi structurées

La syntaxe de base de XML, est parfaitement capable de décrire les données semi structurées ***[GAR 99a]***. Les transformations entre des *ssd-Expressions* de et vers des *documents XML* peuvent facilement être automatisées ***[ABI 00]***. On peut voir dans l'exemple de la figure 4.12, un document XML en (a), et l'expression semi structurée qui lui correspond (b).

```
< Personne >
   <Nom>  Khaled < / Nom>
   <Age >  42 < / Age>
   < Email >  agb@abc.com < / Email >
 < / Personne >
```

(A)

```
{Personne :
 { Nom : « Khaled »,
Age : « 42 »,
Email : « agb@abc.com  »  }
}
```

(B)

Fig. 4.12 – Equivalence entre XML et ssd-expressions

Malgré les similitudes entre les documents XML et les données semi structurées , il existe dans XML certaines notions qui n'ont aucune utilité pour l'échange de données, comme les commentaires , le traitements des instructions (*PI Processing Instruction)* et les *CDATA* ou les entités. ***[ABI 00]***

Enfin , les éléments XML sont ordonnés, alors que les données semi structurées ne le sont pas. Cependant, cette propriété est quasiment inutilisée, puisque pour l'échange de données, Les applications utilisant XML ignorent totalement l'ordre des données.

XML est à présent le format standard utilisé pour présenter des données semi structurées, ainsi les projets utilisant OEM peuvent migrer aisément vers XML. ***[ABI 00], [Gar 00].***

4.1 Le modèle Graphique XML

Alors qu'une *ssd-expression* dénote un graphe dont les arcs sont labellisés, XML , dénote un graphe dont le nœuds sont labellisés . ***[ABI 00], [Gar 00].*** La figure 4.13 illustre la représentation graphique de l'exemple cité ci dessus, dans le modèle XML et semi structuré.

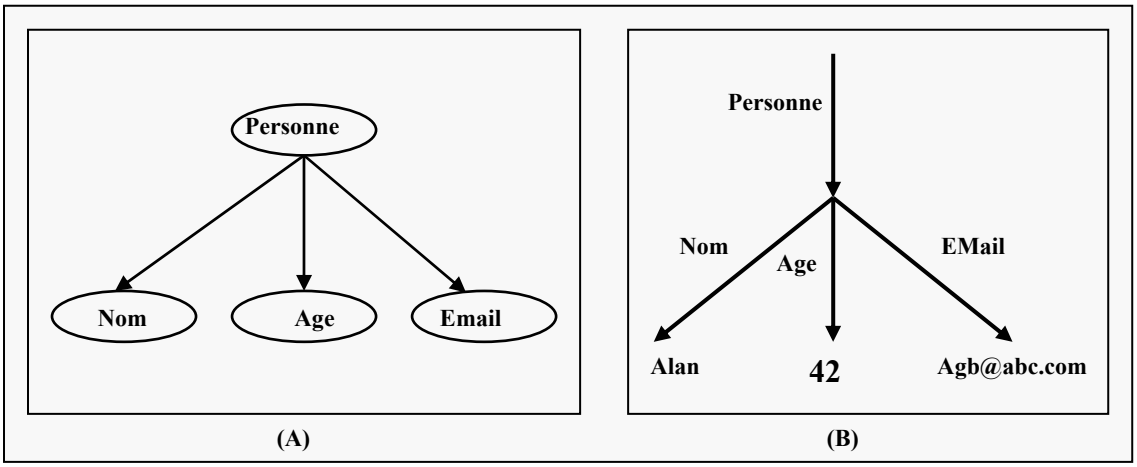

FIG. 4.13 - Arbres de données XML (A) et des SSD-Expressions (B)

4.2 Les références XML

Les données semi structurées sont souvent dénotées par des graphes, plutôt que des simples arbres ; à cause de l'existence de cycle dans les données. Pour résoudre ce problème, XML utilise la technique de *référencement*, en associant un identifiant unique aux éléments, comme valeur d'un attribut qu'on appelle *ID* ***[ABI 00].*** Dans l'exemple de la figure 4.14,on associe l'identifiant *S2* avec l'élément <State> (figure 4.14, (a)) ; on peut ainsi, faire référence à cet élément, en utilisant l'attribut *IdRef*.(figure 4.14, (b)) .

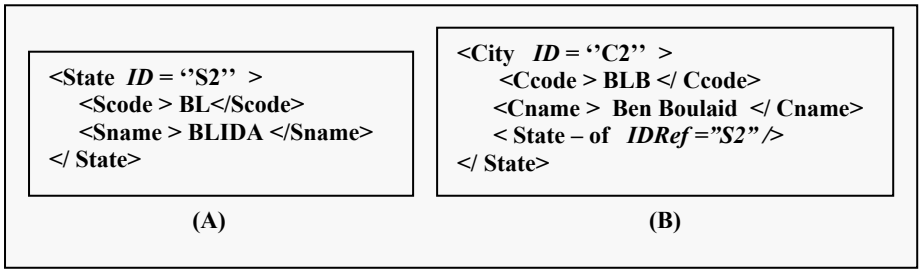

FIG. 4.14 le référencement en XML

5. Méta données pour le modèle XML

Généralement une grammaire définit la syntaxe d'un langage. En XML, une grammaire définit la liste des balises qui sont utilisables, la syntaxe correspond à l'organisation de ces balises. Par conséquent, un document XML est compréhensible si sa grammaire est définie.

Du point de vue des bases de données, une grammaire pour les document XML représente une métadonnée pour les données semi structurées. En effet, le schéma est défini dans les données, et aucune structure n'est précisée à priori ***[GAR 03a]***.Cependant, deux approches ont été étudiées et proposées afin de décrire de telles données ***[DAN 03a]*** **:**

- *Un schéma défini à posteriori :* ce schéma est calculé automatiquement à partir des données. Il est rigide et décrit les données de façon très précise. Toutefois il doit être mis à jour régulièrement. On peut citer l'exemple des *guides de Données (DataGuides).*

- *Un schéma défini à priori :* c'est un schéma flexible qui permet de décrire les données de manière suffisamment ouverte, afin de pallier aux problèmes liés aux structures irrégulières des données. On peut citer comme exemple les XML-Schémas et DTDs.

5.1 Les Guides de Données

Les guides des données (*dataguides*) ont été introduits la première fois ,dans le projet LORE ***[ABI 00]***. Ils résument un graphe de données en *factorisant* les chemins communs à *posteriori.* A partir d'un graphe de données semi structurées (OEM par exemple), on construit un nouveau graphe *le Guide de données,* en respectant les règles suivantes : ***[ABI 00]***

- Chaque chemin dans le graphe de données existe dans le guide de données, l'inverse est vrai aussi.
- Chaque chemin dans le guide de données existe une seule fois.

Pour créer *un Guide de données* , on exécute l'algorithme suivant : ***[ABI 00]***

1. Création de nœud racine appelé *Root.*
2. A chaque chemin dans le graphe de données ,créer un nœud dans le *Guide de données.*

L'exemple de la figure 4.15 illustre le guide de données (d) des données en (a), (b) et (c).

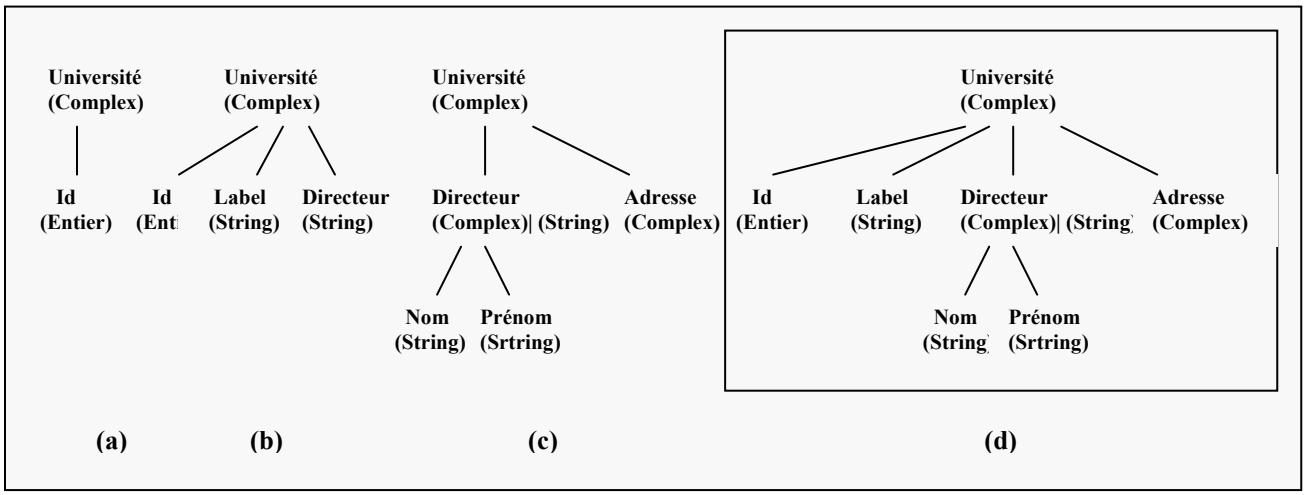

FIG. 4.15 - Le guide de données

5.2 La DTD *(Document Type Definition)*

Une DTD (*document type définition)* est un ensemble de règles, que les données d'un document XML doivent respecter pour que celui ci soit valide. La DTD est décrite par un ensemble de déclarations DOCTYPE, qui comportent les noms des éléments ainsi que leurs structures hiérarchiques. ***[STR 00]***

Les DTDs peuvent êtres utilisées comme des schémas de données semi structurées***[LUD 99]***. Cependant , la DTD permet uniquement de décrire la structure d'un document XML (liste des balises et organisations des balises), et non le typage des données contenues (chaîne de caractère, date, entier, …etc.). Dans la figure 4.16, le document XML (a) est valide par rapport à la DTD (b) :

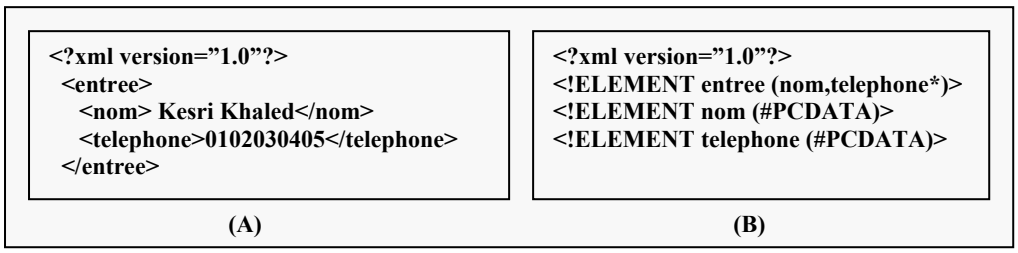

FIG. 4.16 – Document XML et sa DTD

- La cardinalité des données est précisée en utilisant les symboles : '+' (le sous élément est représenté une fois au moins) ; '*' (le sous élément est représenté zéro ou plusieurs fois) ; ' ?' (le sous élément est optionnel) ; sans symbole (le sous élément est représenté exactement une fois). ***[GIR 01]***

- Par contre, on ne peut pas typer les données , seulement pour les chaîne de caractères où on notera #PCDATA ou #CDATA ; pour décrire une chaîne de caractères *parsée* (texte contenu dans un élément) ou pas (Texte contenu dans les valeurs d'attributs). ***[GIR 01].***

5.2.1 Référencement à Une DTD

On peut référencer une DTD de deux manières suivantes : ***[GIR 01]***

- *Interne :* en définissant la DTD dans le corps du document XML.On utilise l'élément *<!DOCTYPE nom_grammaire [définitions]>*. Cette solution est rarement utilisée.

- *Externe :* Un document XML peut préciser ,dans son entête , qu'il respecte une DTD définie dans un document à part. On fait référence alors, à la DTD par son nom et son URL.

5.2.2 Les constituants d'une DTD

Une DTD définit une structure de document, en utilisant un ensemble de déclarations de balisage. Une déclaration de balisage peut être l'une des : ***[GIR 01]***

- *Déclaration de type d'élément :* Les éléments XML sont définis dans une balise <!ELEMENT>. La syntaxe de base est <!ELEMENT nom (contenu)>. Un élément peut être vide, contenir une chaîne de caractères, ou contenir des sous éléments.

- *Déclaration d'attributs :* Les attributs sont tous décrits dans une balise <!ATTLIST>. La syntaxe de base est :

```
<!ATTLIST
nom-element nom-attribut1 type valeur-par-defaut #REQUIRED
nom-element nom-attribut2 type valeur-par-defaut #REQUIRED >
```

  Chaque attribut de la liste est suivi d'un critère existentiel (#REQUIRED si l'attribut est obligatoire , #FIXED s'il est fixé, #IMPLIED s'il n'a pas de valeur par défaut).

- *Déclaration d'entité :* Les entités sont des variables qui peuvent être utilisées dans tout document, et ceci en précédant le nom de l'entité par le symbole '&'. Leur déclaration est contenue dans l'élément <!ENTITY>.

- *Les Identifiants :* Un attribut de type ID, permet d'attribuer un identifiant à un élément d'un document XML. Cet élément peut être ensuite référencé dans une autre partie du même document en utilisant un attribut de type IDREF.

5.3 Les Schémas XML

Bien que les DTDs soient utiles pour les applications web, elles présentent les limites suivantes: ***[STR 00]***

- Les DTDs ne permettent pas la spécification de types de données comme les nombres et les dates.
- Les DTDs ne peuvent être réutilisées ou combinées, pour créer de nouvelles définitions de données.
- Les DTDs sont écrites dans une syntaxe particulière qui leur est propre, donc leur interprétation peut être ardue pour des utilisateurs débutants.
- Il est difficile d'étendre les fonctionnalités des DTDs.

Les schémas XML-schémas sont la solution préconisée par Microsoft et standardisée par W3C, afin de combler les limites des DTDs. Un schéma est une spécification formelle des règles d'un document XML : il décrit quels sont les éléments autorisés (permis) dans le document, et dans quelle combinaison peuvent t-ils être utilisés. Les schémas fournissent les mêmes fonctionnalités que les DTDs ; Cependant, parce qu'ils sont décrits en XML , il sont extensibles. On peut donc augmenter les schémas avec des informations additionnelles ,telles que les types des données, l'héritage, et les règles de présentation.

En pratique les schémas permettent aux auteurs des pages web de : ***[GIR 01]***

- Définir précisément quels sont les noms des éléments permis dans le document et , dans chaque élément, leurs éléments fils, leurs attributs, ainsi que les relations père – fils qui sont permises.
- D'importer des fragments de codes de d'autres schémas.

- D'étendre les types à travers l'héritage. Ceci permet des relations complexes entre les éléments, tout en gardant la structure lexicale simpliste d'arbre .

Dans l'exemple suivant nous pouvons préciser le type des informations contenues dans les éléments <nom> (*string*) et <téléphone> (*décimal*) de l'élément <entrée> du document (a) ; à l'aide de XML-Schema décrit en (b) .(figure 4.17)

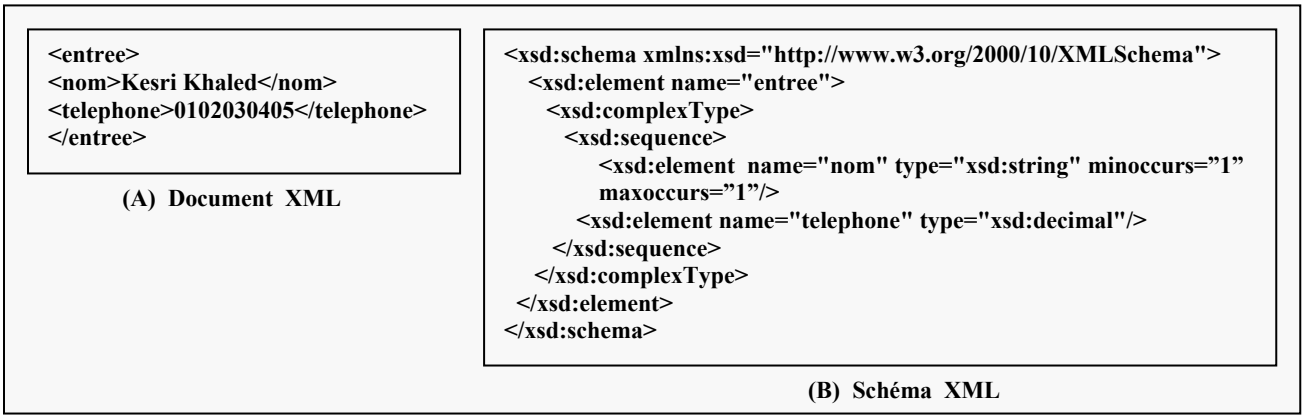

FIG. 4.17 - Document XML et son schéma XML-schéma

Un schéma XML-Schema permet de *typer* un document XML, dans un formalisme proche des bases de données. Il permet ainsi de typer une requête, et de vérifier sa validité avant exécution.***[DAN 03a]*** prenons l'exemple de la requête suivante, qui affiche tous les titres des entrées qui ont un attribut *zip* avec une valeur « 12345 » ***[ABI 00]* :**

```
Select x.title
From  biblio._ x
Where x.*.zip = ''12345''
```

Pour évaluer cette requête, on vérifiera tous les objets Biblio , ainsi que leurs sous éléments, pour chercher un attribut *zip* avec la valeur requise. Par contre, si on formule cette requête sur un XML-schéma ,l'analyseur de la requête va typer à l'aide de ce schéma, c'est à dire , qu'il va déduire les différents domaines de valeurs utilisés ;ce qui réduira l'espace de recherche. On pourra savoir par exemple, que seules les entrées *Book* contiennent un attribut *Zip.* Ou plus encore, que l'attribut *Zip* peut apparaître seulement derrière l'attribut *Adresse.* Donc une optimisation de la requête précédente serait :

```
Select X.title
From  biblio.book X
Where X.adresse.zip =”12345”
```

5.3.1 Référencer un XML-Schema

Un document XML peut préciser qu'il respecte un XML-Schema. Cela se fait dans l'entête de ce document ***[ABI 00]*** ; dans l'exemple suivant, L'espace de noms *xsi* ,correspond aux instances de document XML respectant les contraintes définies dans un document XML-Schema.

```
<entree xmlns="http://www.annuaire.org"
xmlns:xsi="http://www.w3.org/2000/10/XMLSchema-instance"
xsi:schemaLocation="http://www.annuaire.org/entree.xsd">
<nom>Kesri Khaled</nom>
<telephone>0102030405</telephone>
</entree>
```

5.3.2 Les Constituants d'un XML-Schema

Un schéma XML-schéma est un document XML, qui permet de définire la syntaxe d'un document XML de données ; en utilisant des éléments XML prédéfinis, il déclare la syntaxe du document de données. Les constituants de base d'un schéma XML-schéma sont : ***[ABI 00]*** .

- La déclaration du schéma.
- Les déclarations de types de données.
- Les déclarations d'éléments.
- Les déclarations d'attributs.
- Les déclarations d'identifiants.

Déclaration du schéma XML-schéma

L'élément *<xsd:schema>* permet de déclarer un document XML-Schema. Son attribut *targetNamespace* permet de préciser l'espace de nom de ce type de documents. L'attribut *elementFormDefault* précise si les documents XML respectant cette grammaire doivent référer à cet espace de nom , Exemple :

```
<xsd:schema  xmlns:xsd="http://www.w3.org/2000/10/XMLSchema"
targetNamespace="http://www.annuaire.org"
xmlns="http://www.annuaire.org"
elementFormDefault="qualified"/>
```

Déclaration de types de données

Le standard XML-schéma définit plus de 40 types de données, parmi lesquels on retrouve *string*, *integer*, *date*, *year*, *CDATA*, *float*, *double*, *binary*, *ENTITIES*, *token*, byte,…etc. Il fournit aussi un mécanisme de définition de types de données complexes. La balise *<xsd:complexType>* est alors utilisée. Dans l'exemple qui suit, le type complexe *TypeEntree* est défini comme suit :

```
<xsd:complexType name="typeEntree">
  <xsd:sequence>
    <xsd:element name="nom" type="xsd:string"/>
     <xsd:element name="telephone" type="xsd:decimal"/>
    </xsd:sequence>
</xsd:complexType>
```

On déclare l'élément comme suit :

```
<xsd:element name=”entree” type=”typeEntree”>
```

Ce qui permet de définir le document comme suit :

```
<entree>
  <nom>string</nom>
   <telephone>decimal</telephone>
</entree>
```

Les schéma XML-schémas permettent de définir des nouveaux types de données, à partir de types existants. Pour ce faire, on utilise les deux techniques suivantes :

- *Restriction de type existant (<xsd:simpleType>) :*on peut créer un nouveau type de données, en restreignant un type de *base* à certaines valeurs. Dans l'exemple qui suit , on désire définir un type basé sur le type *string* , qui représente un choix (oui/non). Ce type appelé *ChoixOuiNon.* on déclare un élément *choix* comme suit *:*

```
<xsd:simpleType name="choixOuiNon">
      <xsd:restriction base="xsd:string">
        <xsd:enumeration value=”oui”/>
        <xsd:enumeration value=”non”/>
      </xsd:restriction>
</xsd:simpleType>
<xsd:element name=”choix” type=”choixOuiNon”/>
```

Une instance de cet élément est :

```
<choix>oui</choix>
```

- *Extension d'un type de données existant (<xsd:complexContent>) :* Il est aussi possible de créer des types dérivés, en ajoutant des éléments aux structures déjà créées. Par exemple, dans le type *'entree'* défini précédemment, on peut créer un type dérivé : *entreeAvecAdresse*, contenant en plus du nom et du numéro de téléphone, une adresse de type string .

```
<xsd:complexType name="entreeAvecAdresse">
    <xsd:complexContent>
      <xsd:extension base="entree" >
        <xsd:sequence>
        <xsd:element  name="adresse" type="xsd:string"/>
        </xsd:sequence>
      </xsd:extension>
    </xsd:complexContent>
</xsd:complexType>
```

Déclaration des éléments

Pour définir un élément, on utilise la balise *<xsd:element>*. La cardinalité des éléments est gérée dans les XML-schemas en utilisant les attributs *minoccurs* et *maxoccurs* de l'élément *<element>*. La définition d'un élément procède des deux manières suivantes

1. Définir un type de données , et l'utiliser dans la définition de l'élément. Dans ce cas, on peut créer des types réutilisables dans le reste du document.

```
<xsd:complexType name="entree">
    <xsd:sequence>
      <xsd:element name="nom" type="xsd:string"/>
      <xsd:element name="telephone" type="xsd:decimal"/>
    </xsd:sequence>
</xsd:complexType>
```

2. Définir le type de données à l'intérieur de l'élément.

```
<xsd:element name="entree">
<xsd:complexType>
    <xsd:sequence>
      <xsd:element name="nom" type="xsd:string"/>
      <xsd:element name="telephone" type="xsd:decimal"/>
    </xsd:sequence>
</xsd:complexType>
</xsd:element>
```

Déclaration des Attributs

La définition des attributs associés à un élément se fait dans un élément *<xsd:attribute>*. Chaque élément *<xsd:attribute>* possède les attributs suivants :

- name : nom de l'attribut.
- type : type de l'attribut. Par exemple xsd:string, xsd:boolean,...etc.
- use : précise si l'attribut est obligatoire (required) ou optionnel (implied), ou valeur fixée (Fixed) ou par défault.
- value : valeur par défaut de l'attribut.

L'exemple suivant décrit une définition de l'élément *Elément ,* avec l'attribut *att* optionnel, et dont la valeur par défaut est « a » ; et l''attribut *at2* est obligatoire et a comme valeur par défaut « true ».

```
<xsd:element name="element">
  <xsd:complexType>
    <xsd:attribute name="att" type="xsd:string" use="implied" value="a"/>
    <xsd:attribute name="at2" type="xsd:boolean" use="required" value="true"/>
    </xsd:complexType>
</xsd:element>
```

Une instance de cet élément est décrite comme suit :

```
<element att=”hello” at2=”true”/>.
```

Déclaration des Identifiants

Comme pour les DTDs, il est possible d'attribuer un identifiant aux éléments, en définissant un attribut de type ID. Cet attribut , permet de les référencer dans la suite du document XML, en utilisant un attribut de type IDREF. Dans l'exemple suivant, on définit un élément possédant un attribut identifiant comme suit :

```
<xsd:element name=”element1”>
    <xsd:complexType>
      <xsd:attribute name=”id” type=”ID” use=”required”/>
    </xsd:complexType>
</xsd:element>
```

Ensuite, on définit un élément avec un attribut de type *IDREF*, qui lui fait référence. :

```
<xsd:element name=”element2”>
    <xsd:complexType>
      <xsd:attribute name=”ref” type=”IDREF” use=”required”/>
    </xsd:complexType>
</xsd:element>
```

Une instance des deux éléments , où l'élément <element2> fait référence à l'élément <element1> est décrite comme suit:

```
<element1 id=”A”/>
<element2 ref=”A”/>
```

6. Manipulation d'un document XML

Pour accéder à un document XML, trois approches existent : ***[ABI 00] [STR 00]***

6.1 Les analyseurs type « Callaback »

Ils sont basés sur un système d'événements. En parcourant le document XML, l'analyseur renvoie un événement à chaque élément rencontré. Le programmeur définit le traitement à effectuer pour chaque événement, et peut ainsi construire sa propre structure de données. Cette approche n'est cependant pas adaptée lorsqu'on désire modifier le document XML. SAX (Simple API For XML) est l'API la plus connue se basant sur cette technique. Elle est indépendante du langage de programmation.

SAX (*Simple API For XML*) est une API pour la manipulation de documents XML ne prenant pas en compte la structure d'arbre d'un document. Des événements sont renvoyés à chaque fois que l'analyseur rencontre une nouvelle balise. Le programmeur définit le traitement à effectuer en implémentant un « handler » (ou « gestionnaire d'événements ») qui

récupère les événements. L'avantage de cette méthode est le gain de mémoire par rapport à un analyseur de type DOM qui construit un arbre en mémoire.

6.2 Les analyseurs de type « arbre »

Utilisent la structure arborescente d'un document XML. L'analyseur charge le document XML en mémoire, sous la forme d'un arbre. DOM (Document Objet Model) est l'API la plus connue répondant à cette approche. L'utilisateur parcourt et modifie l'arbre à l'aide des interfaces définies par le W3C. Cette API est aussi indépendante du langage de programmation ;

DOM *(Document Object Model)* est une initiative du W3C. C'est une API pour la manipulation de documents XML. elle permet de représenter un document XML sous la forme d'un arbre chargé en mémoire, et fournit les interfaces permettant de le manipuler. Les interfaces DOM sont indépendantes de tout langage de programmation. Les implémentations de DOM utilisent des analyseurs SAX pour charger l'arbre en mémoire.

6.3 Le DataBinding

Cette méthode est une spécification Java de Sun, qui permet de charger un document XML en mémoire sous la forme d'un objet, que l'on manipule en utilisant des accesseurs. On sauvegarde ensuite le document en sérialisant l'objet en XML.

7. Conclusion

Les données semi structurées sont caractérisées par une structure irrégulière, dynamique ou inconnue ; néanmoins, elle est implicite, puisque la description des données est incluse avec celles ci, d'où le terme de données *auto descriptives*.

Par conséquent, les données semi structurées permettent de décrire efficacement, les données qui ne peuvent pas avoir de modèle conventionnel (Relationnel, Objet). Ils sont donc, le moyen idéal, pour l'intégration des données. En effet, Intégrer plusieurs données hétérogènes nécessitent l'utilisation d'un modèle de données commun, suffisamment *dynamique* , pour permettre d'exprimer les irrégularités engendrées par l'hétérogénéité des données.

XML (*Extensible Markup Language*) a émergé comme langage d'échange de données sur le web. Il s'est vite imposé comme modèle des données semi structurées. Dans la plupart des travaux dans ces domaines le modèle de données commun qui a été choisi est XML. En effet, il présente les avantages suivants :

- Un modèle complet *:* les schémas XML peuvent être utilisés comme des schémas de conception complets, permettant d'exprimer les relations, les attributs, l'héritage, le typage, l'hiérarchisation…etc.

- Un schéma d'échange explicite inclus dans les données : possibilité d'unification des noms en utilisant des espaces de noms.

- Une simplicité de génération depuis des formats divers : SQL, fichiers légataires, documents (HTML,…etc.).

De ce fait, la plupart des recherches , actuellement menées, dans le domaine des systèmes de gestions de bases de données fédérées, ont choisi XML-couplé avec un langage de requêtes- comme modèle commun ; pour l'intégration des différents modèles issus de la fédération.

Chapitre 5 :Les Systèmes de Médiation basés XML

1. Introduction

XML est devenu aujourd'hui, le format standard pour l'échange des données sur le web. De ce fait , il est intégré par la plupart des éditeurs de logiciels des bases de données. Il permet entre autre d'exprimer, grâce à son modèle de donnée ,vu dans le chapitre précédent, différents formats de données qu'elles soient structurées (bases de données relationnelles, objet), semi structurées (données du web) ou pas structurées du tout (e-mail , fichier texte,…etc.).

Pour élaborer des systèmes de bases de données fédérées basés sur XML ; il est impératif d'utiliser une architecture capable de prendre en charge les fonctionnalités de tels systèmes ; à savoir, la traduction des modèles, des langages de requête, l'intégration des données, …etc.; de plus, l'utilisation de XML comme modèle commun aux différentes sources hétérogènes du système, nécessite la définition d'un langage de requêtes XML, suffisamment puissant pour pouvoir manipuler les données.

Le but de ce chapitre, est de présenter l'architecture standard, ainsi que les fonctions de base des systèmes de bases de données fédérées basés sur XML.

Dans la section (2), on introduit les systèmes de médiation; à la section (3), on en explique les principaux notions. La section (4) développe l'architecture de référence de ces systèmes. Les spécificités d'un langage de requête pour données XML sont décrites dans la section (5). La section (6) présente un ensemble de langages de requêtes pour données XML. le langage XQuery , vu son importance dans notre travail est étudié à part dans la section (7). Dans la section (8) , on présente en détail quelques architectures de médiation existantes. On conclut à la section (9).

2. L'approche systèmes de médiation

Pour implanter le concept des bases de données fédérées vu dans la chapitre (3), plusieurs approches ont été utilisées , parmi lesquelles l'approche entrepôt de données, ou celle basée sur les *systèmes de médiation****[GAR 02a]***. Dans ce cas, les données intégrées ne sont pas matérialisées dans un SGBD. L'intégration se fait sur demande pour répondre aux requêtes.***[GAR 03a]***

Concevoir une base de données fédérée au dessus d'une couche de données hétérogènes et pré-existantes, implique la prise en considération de deux éléments majeurs suivant :

- *Un modèle de donnée commun (Figure 5.1):* en effet, dans un environnement hétérogène , les modèles de données peuvent être différents, peuvent se chevaucher (cas de redondances des données) ou autres. D'où la nécessiter d'avoir un modèle de données commun ,afin de pouvoir avoir une vue globale, cohérente sur l'ensemble des données existantes. ***[GAR 03a]***

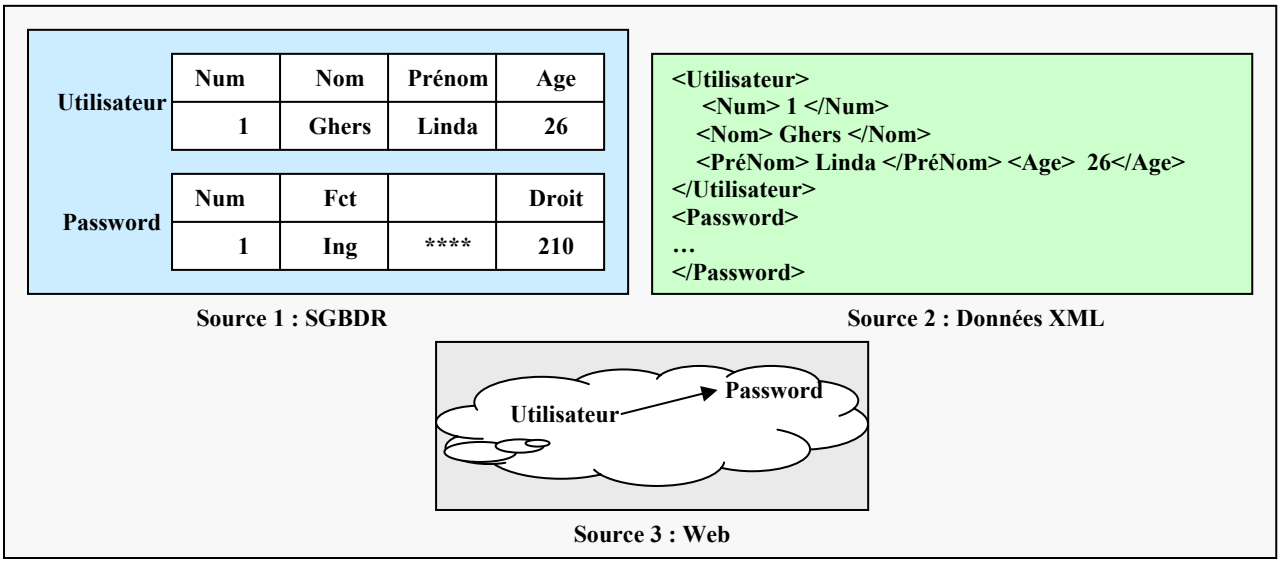

FIG. 5.1- Différents modèles de données existants

- *Un langage d'interrogation commun (Figure 5.2):* afin que l'utilisateur ait l'illusion d'un accès à une seule base de données virtuelle, il doit pouvoir utiliser un langage de requête unique ,et retourner à l'utilisateur un résultat sous format lui aussi unique ***[DAN 03a]***

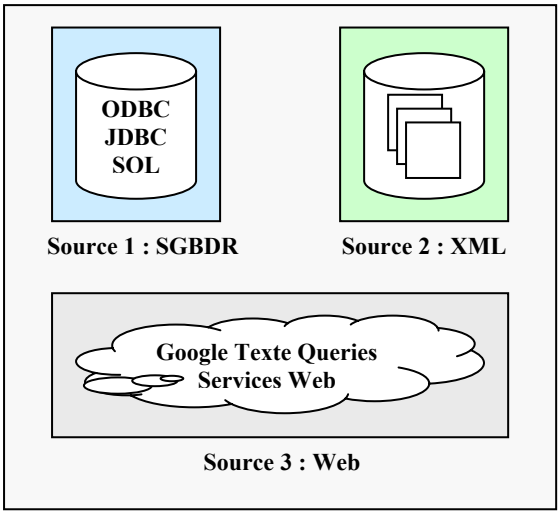

FIG. 5.2 - Différents Langages de requêtes

3. Concepts de base

Dans un système de médiation, l'utilisateur traite avec une vue intégrée des données ; sur laquelle, il peut formuler des requêtes globales. Lors de la conception de tels systèmes on doit assurer les mécanismes suivants :

- *Intégration des schémas :* l'intégration des schémas consiste dans l'élaboration d'un schéma global, qui représente une vue homogène sur les schémas des différentes sources de données sous jacentes.

- *Intégration des données :* consiste dans l'intégration des données issues des différentes sources lors de la composition des résultats des requêtes.
- *Evaluation des requêtes :* consiste dans la décomposition d'une requête globale en sous requêtes , l'évaluation effective de celles ci, puis dans la recomposition des résultats.

Lors de l'intégration des données et des schémas, plusieurs anomalies peuvent surgir: ***[DAN 03a]***

- *Redondance des données :* lorsqu'une même donnée survient sur plusieurs sites.
- *Correspondances des noms :* et ceci des deux manières suivantes :
  - *Même nom d'entité, mais significations différentes :* par exemple ,une même entité désignant la Nationalité d'une personne, peut être appelé *nationalité* dans un site, et *Pays* dans un autre.
  - *Noms différents mais signification identique :* sur un site l'entité *année* peut signifier l'année d'inscription d'un étudiant, sur un autre ,ceci signifie l'année en cours.
- *Conflits de schémas :*quand la même entité est modélisée de manière différentes selon le site, *nationalité* dans un site peut comporter les occurrences « Algérien, Nigérien, Marocain » et dans un autre site elle peut comporter les occurrences : « Alg, Nig ,Mar »

4. Architecture de référence I3

Les bases de données fédérées, ou *systèmes à médiation***,** sont utilisées pour l'intégration des données, où la fraîcheur des données est un besoin critique. L'élaboration de tels systèmes nécessite une architecture capable de faire communiquer les différentes sources de données mises en œuvre. Les architectures des base de données fédérées, ont convergé vers un standard proposé par DARPA et Gio Widerhold I3 ***[WG 93] [GAR 02a] [GAR 99b][GAR 03a]***: (figure 5.3). Cette architecture s'articule en trois niveaux***[WG 93]*** :

- *Niveau Client (présentation):* ce niveau comporte les *applications clientes*.
- *Niveau Médiateur :* Il est formé de *médiateurs* chargés de l'intégration des données en provenances des différentes sources. Ce niveau est composé aussi de *facilitateurs* qui permettent d'identifier les sources et de composer les réponses aux utilisateurs.
- *Niveau Source (Adaptation) :*Il est composé d'*adaptateurs* (*wrappers)* chargés de transformer les données locales, de sorte à les rendre assimilables par le médiateur ; et aussi de filtrer autant que possible les données suites au requêtes. Un adaptateur accepte une requête donnée dans le langage commun du médiateur, la transcrit dans le langage natif de la source, et exécute la requête. Le résultat de la requête est alors transformé, suivant le modèle de donnée global du médiateur ,et renvoyé à celui ci.

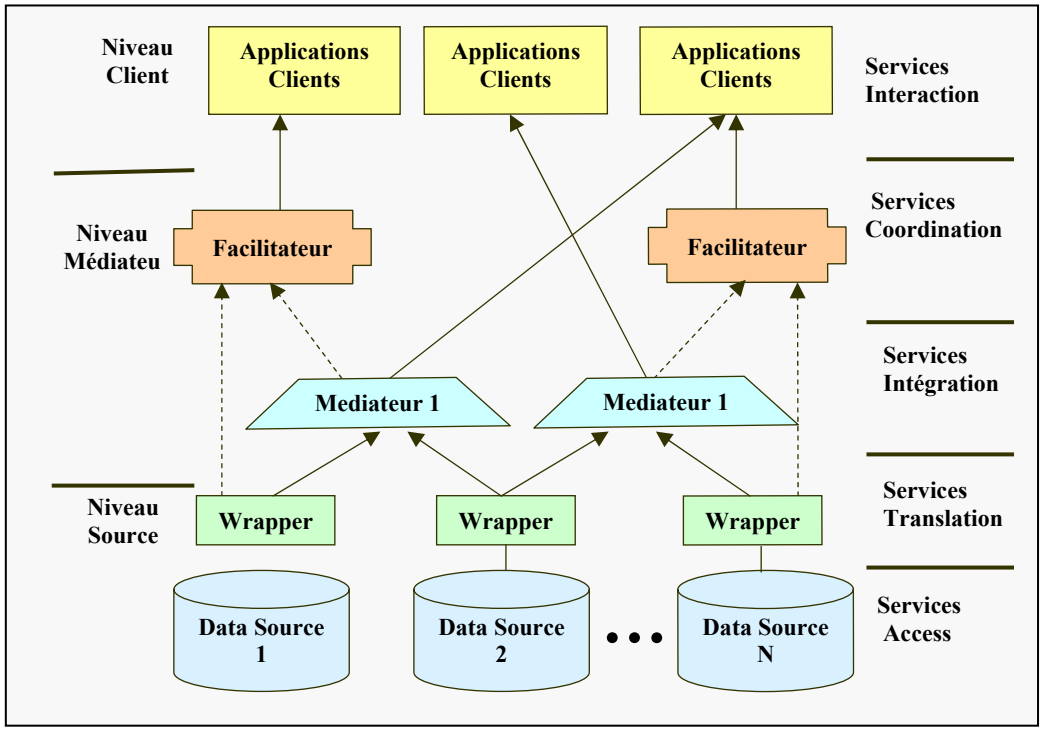

FIG. 5.3 - L'architecture DARPA I 3

Dans une architecture de médiation donnée, un langage et un modèle commun entre les différents composants à savoir, *les facilitateurs, les médiateurs* et *les adaptateurs,* doivent être définis afin de leur permettre l'échange d'informations et de données. En effet, quand un utilisateur envoie une requête globale au système de médiation, le *facilitateur* la reçoit, il identifie les différentes sources de données mises en cause dans la requête, qui peuvent être des sources de données ou des *médiateurs*. Quand un *médiateur* reçoit une requête globale ,il la décompose en sous requêtes, qu'il envoie aux *adaptateurs* concernés. Chaque *adaptateur* transforme la requête écrite dans le modèle commun, en une requête écrite dans le modèle natif de la source qui lui est associée. La requête est ainsi exécutée au niveau de chaque source ; enfin l'*adaptateur* transforme le résultat dans le modèle commun ,et renvoie le résultat au médiateur. Celui ci recompose le résultat global et le renvoie au *facilitateur* qui va la reformater pour la sortie.

5. Un langage de requêtes pour les données XML

5.1 Spécificités d'un langages de requêtes pour XML

Dans le chapitre (4), on a conclu que seul XML est suffisamment puissant et flexible, pour être utilisé comme modèle commun dans une architecture de bases de données fédérées. Cela dit, manipuler des données ainsi modélisées , nécessite l'utilisation d'un langage de requête spécifique.

Un langage de manipulation de données XML doit être , comme son homologue SQL, capable d'utiliser des prédicats complexes sur les données, d'exécuter des jointures, ainsi que de la restructuration des données. Un tel langage doit avoir , entre autre les propriétés suivantes ***[ABI 00]*** **:**

- *Puissance d'expression :* il doit être capable de capturer la sémantique incluse dans les données semi structurées, en définissant une algèbre suffisamment expressive à base d'opérateurs standards de requête de bases de données (projections, jointure…etc.)
- *Possibilité de composer :* un résultat en sortie d'une requête doit être utilisable comme entrée à une autre requête. Ce qui s'avère très important pour la construction des vues.
- *Le schéma :* un langage de requête doit être capable d'exploiter un schéma de données XML, pour vérification de type, optimisation…etc.
- *Manipulation par programme :* pouvoir être généré automatiquement.
- *La construction de résultat :* pouvoir spécifier la façon dont le résultat devra être présenté.

Plusieurs langages répondant plus au moins à ces critères, on été proposés pour permettre l'interrogation des données XML. Deux approches ont dominé la recherche ***[GIR 01]*** :

- *Les langages de requetes simples :* ils se basent sur la structure du document XML, les plus célèbres sont XQL et XPath ***[GIR 01]***. Une requête ressemble globalement à l'adressage dans le système de fichiers. Ils ne permettent pas de construire de nouveaux documents XML, mais uniquement de les consulter.

- *Les langages de requêtes ' type SQL' :* ils permettent de construire des requêtes similaires aux requêtes SQL (SELECT…FROM…WHERE) adaptées à la structure des documents XML.Une requête de ce type permet de générer de nouveaux documents XML. Parmi ces langages on trouve XML-QL***[LES 98]***, Quilt ***[CHA 00]***, XQuery***[ROB 04]***, Lorel et YaTL ***[ABI 00]***. (tableau 5.1).

Langage	Exemple de requêtes
Lorel	**SELECT E** **FROM annuaire.entree E** **WHERE E.nom = "Kesri khaled"**
XML-QL	**WHERE** **<annuaire>** **<entree>** **<nom> $nom </nom>** **</entree>** **</annuaire> ELEMENT_AS $entree** **IN annuaire.XML** **$ nom = ''Kesri khaled''** **CONSTRUCT $entree**
Quilt **XQuery**	**FOR $entree IN** **document("annuaire.xml")/annuaire/entree** **$nom IN $entree/nom** **WHERE $ nom = ''Kesri khaled''** **RETURN $entree**
YATL	**MAKE** **Entree [nom [$ nom] , telephone [$tel]]** **MATCH "annuaire.xml" WITH** **Entree [nom [$ nom] , telephone [$tel]]** **WHERE** **$ nom = ''Kesri khaled''**

Tableau 5.1 - Exemples de langages de requêtes XML

5.2 Les expressions de Chemins (*Path exprssions*)

Une différence majeure entre un langage de requêtes sur modèle XML ,et les langages de requêtes sur modèle structuré est que le modèle XML, représente les données sous forme de graphe (voir chapitre 4). Par conséquent, pour qu'un langage de requête puisse accéder aux données incluses dans de telles structures, il est impératif qu'il soit capable d'atteindre une profondeur quelconque du graphe de données.

Pour ce faire, les langages de requêtes sur XML exploitent la notion des *Expressions de chemins* ***[ABI 00].*** Les expressions de chemins dépendent du modèle sous jacent au langage de requête. La syntaxe utilisée pour les expressions de chemin ,diffère alors d'un langage à un autre ; ainsi, les 'slache' utilisés dans *XQuery* et *XPath* sont remplacés par des **.** (point) dans *OQL*, par exemple. Dans ce chapitre nous utiliserons la syntaxe utilisée par XQuery. ***[ROB 04]***

Les expressions de chemins servent surtout à localiser un nœud en identifiant son emplacement dans le graphe. Une expression de chemin consiste dans une série de un ou plusieurs *pas (*steps*)* **,** séparés par des symboles (exemple : /). Chaque *pas* évalue une séquence de nœuds ***[ROB 04] [ROB 03].***

Exemple : Pour accéder à la liste des nœuds « nom » de l'annuaire suivant :

```
<annuaire type="pages blanches">
  <entree>
    <nom>Kesri Khaled</nom>
    <telephone>06 03 02 01 00</telephone>
     <profession>  Ingénieur réseau  </profession>
  </entree>
  <entree>
    <nom>Toumi  Karim</nom>
    <telephone>06 00 01 02 03</telephone>
     <profession>  Dentiste  </profession>
  </entree>
</annuaire>
```

On formule la requête d'expression de chemin suivante :

```
/annuaire/entree/nom
```

La liste des nœuds répondant à cette requête est décrite comme suit :

```
<nom>Kesri Khaled</nom>
<nom>Toumi Karim</nom>
```

L'évaluation des *expressions des chemins* utilisée dans les langages de requêtes, est basée sur *la recherche par motif* (*pattern matching*) ***[ROB 04].*** Les motifs peuvent être des chemins complets, des expressions régulières ou autres . Parmi les plus importantes propriétés des expressions de chemins, on peut citer les suivants :

- *Les expressions régulières :* en combinant un ensemble de motifs (tableau 5.2) dans des expressions régulières ; les expressions de chemins permettent d'exprimer des chemins qui ont des propriétés communes (exp. le chemin doit traverser le nœud *entrée*). Ou d'exprimer une propriété propre au nœud (exp. : le nom du texte doit contenir une chaîne de caractère « BIB »). Exemple : l'expression *annuaire.entree.(nom | téléphone)* sera liée au nœud *nom* ou au nœud *téléphone*. ***[ABI 00].***

Symbole	Rôle
\\ (XPath) ou _ . (OQL)	Ce symbole se lie à n'importe quel chemin. Exe : annuaire\\entree : sera évalué (lié) à tous les chemins commençant par annuaire et se terminant par entree.
* (Kleen)	Ce symbole spécifie un nombre d'occurrences arbitraire d'une expressions régulière.
+	Ce symbole spécifie Une ou plusieurs occurrences.
?	Ce symbole spécifie zéro ou une occurrence de l'expression régulière (chemin optionnel).
|	Disjonction

Tableau 5.2 – Principaux motifs des expressions régulières

- *Les prédicats :* ce sont des expressions booléennes, mises entre crochets [] ; elles permettent d'effectuer un tri sur des éléments obtenus par une requête simple. On peut effectuer un test sur le nom d'un élément, son contenu, ses attributs, sa position dans l'arbre, etc. Exemple : le résultat de l'évaluation du prédicat *nom = '' **kesri khaled**''* est vrai. Le tableau (tableau 5.3) illustre les symboles utilisés dans les prédicats :

Symbole	Rôle	Exemple
[i]	I éme élément de la sélection	/A / * [4]
[Last ( )]	Dernier élément de le sélection.	/A / * [Last ()]
@ att = ...	Condition sur les attributs.	/ A / B [@ att='ENG']

Tableau 5.3 – Principaux Symboles des Prédicats

6. Langages de requêtes XML

Dans cette section, nous allons aborder les langages de requêtes XML les plus utilisés. Pour illustrer les exemples, on utilise le document décrit dans la figure 5.4 qui représente une base de données des étudiants universitaires. On suppose que ce document est maintenu à l'adresse URL : http://etudiant.ministere.gov

```
<Les Etudiants>

<Etudiant>
    <nom>  Kesri </nom>
    <Prénom> Khaled  </Prénom>
    <Inscription>
        <Etablissement>  USTHB </ Etablissement >
        <Annee> 1995</ Annee >
     <Inscription>
    <Inscription>
        <Etablissement>  UBLIDA </ Etablissement >
        <Annee> 1994</ Annee >
   <Inscription>
</Etudiant>

<Etudiant>
    <nom>  Hammimded </nom>
    <Prénom> Leila  </Prénom>
    <Inscription>
        <Etablissement>  USTHB </ Etablissement >
        <Annee> 1995</ Annee >
     <Inscription>
 </Etudiant>

<Etudiant>
    <nom>  Abdi </nom>
    <Prénom> Ferial  </Prénom>
    <Inscription>
        <Etablissement>  USTHB </ Etablissement >
        <Annee> 1996</ Annee >
     <Inscription>
 </Etudiant>

<Les Etudiants>
```

Fig. 5.4 – Document « Etudiants.xml »

6.1 OEM-QL

Conçu à l'université *Standford* , le langage OEM-QL (*Object Exchange Model query language*) permet d'interroger des graphes OEM. Il est basé sur le concept des expressions de chemin. En effet, il définit une expression de chemin *simple* ,comme une séquence d'étiquettes séparées par des points, décrivant un parcours dans le graphe OEM. OEM-QL étend ensuite cette notion avec les expressions de chemins *généralisés,* en utilisant les motifs à la place des étiquettes.***[DAN 03a]***

La requête OEM-QL suivante répond à la question suivante : « renvoyer le nom de la personne dont l'établissement contient le mot ''US'' ».

```
Select  Etudiant.nom
Where Etudiant. Etablissement ? like  ''%US%''
```

6.2 XPath

XPath ***[GIR 01] [ROB 04]*** est un langage de requête minimal, basé sur les expressions de chemin. Une expression XPath est utilisée comme une requête qui peut être appliquée à un document ou une collection de documents. Son résultat sera un ensemble de sous arbres,

répondant aux critères de l'expression. Ainsi , *XPath* permet d'exprimer des requêtes complexes. Parmi lesquelles on cite :

- *Recherche par contexte :* en utilisant des opérations permettent de rechercher des nœuds en fonction de leur contexte ; c'est à dire leur positionnement dans l'arbre. (figure 5.5 , requête (a)): *descendant* ; *ascendant* ; *child* ; *parent.*

- *Fonctions :* XPath fournit des fonctions prédéfinies ; parmi lesquelles on cite, *texte()* ; *last()* ; *position() ; count*() …ext. Ces fonctions permettent d'écrire des requêtes complexes ; comme rechercher un élément dont le nom contient une chaîne de caractères, par exemple (figure 5.5, requête (b)).

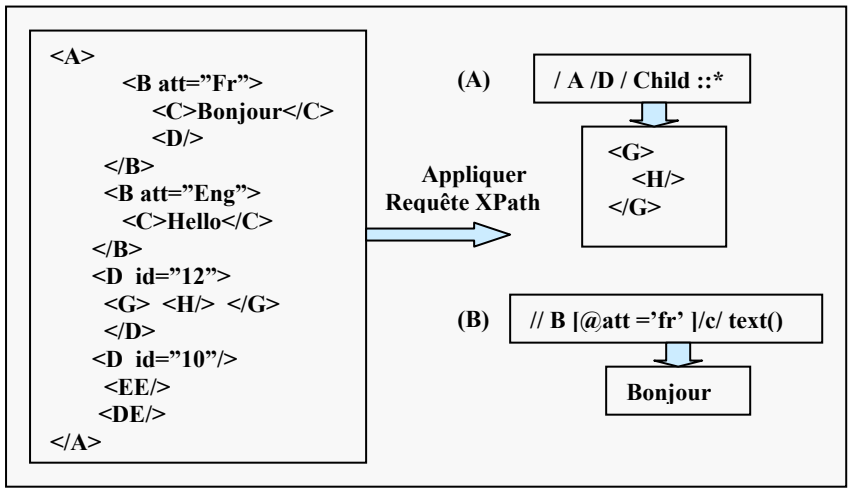

FIG. 5.5 – Exemple de Requêtes XPath .

6.3 XML-QL

Ce langage utilise le modèle des données semi structurées basé sur un graphe à arcs labellisés. Il est basé sur la reconnaissance de motifs (*pattern matching*). Il intègre tous les opérateurs relationnels , pour l'extraction, la conversion et la transformation des données XML ; Il permet aussi , le groupement en utilisant des requêtes imbriquées et les fonctions de skolems ***[LES 98]***. Les données d'un document XML sont alors extraites en utilisant la clause WHERE en spécifiant le modèle des éléments qui doivent être utilisés et en faisant correspondre leurs contenus à des *variables*. ***[DAN 03a][ABI 00]***.

La requête décrite dans l'exemple suivant (figure 5.6) extrait tous les étudiants de l'université de l'USTHB.

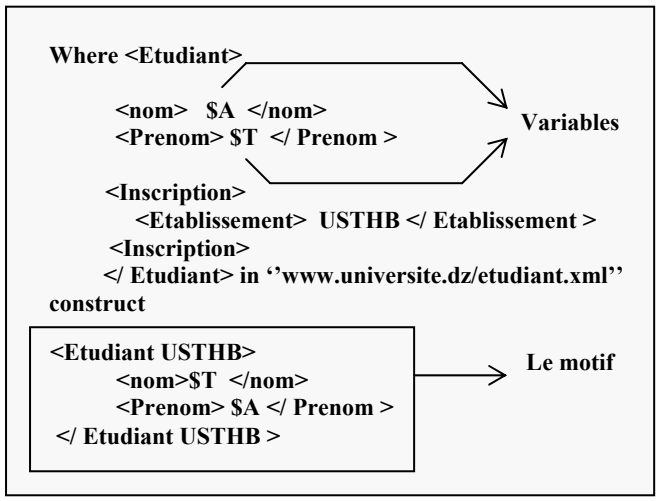

FIG. 5.6 – Requête XML-QL

L'analyseur procède à la reconnaissance du motif à travers le document XML « *etudiant.xml* ». Pour chaque structure du document XML correspondante au motif de la requête, l'analyseur met le contenu de *nom* et *prénom* des éléments, dont l'établissement est USTHB, dans les variables $A et $T. Le mot clef *CONSTRUCT* permet de définir le modèle des données résultantes de la requête.***[ABI 00]***

7. Un support pour le modèle XML : XQuery

XQuery est une spécification réalisée par le groupe de travail *" Query Language Working Group"* du W3C ***[XQ 03] [BOT 03]***. C'est un langage de requêtes sur données XML, basé sur XPath. Il est issu du langage de requête *QUILT* ***[CHA 00].***Il reprend également les avantages de XML-QL ***[BOT 03] [LES 98]*** et XQL,une variante de *XPath* . Dans ce qui suit , on présente les éléments de base , de la syntaxe du langage XQuery. ***[XQ 03]***

7.1 Les fonctions d'entrées (*Input Functions*)

Les fonctions d'entrées sont des fonctions prédéfinies, qui servent à identifier le document dans lequel , la requête XQuery va être évaluée. XQuery utilise les deux fonctions d'entrées suivantes : ***[ROB 04]***

- *doc() :* elle retourne un document entier, plus précisément le nœud racine d'un document identifié par son *URI (Universal Resource Identifier).* Exemple :

```
document( ''Etudiant.xml'')
```

- *Collection() :* elle retourne une collection, qui est la séquence de nœuds associée avec l'*URI*. Elle permet d'identifier une base de données utilisée dans une requête. Exemple :

```
collection( ''Etudiant.xml'')//Etudiant/nom/text()
```

7.2 Création de nœuds

Dans une requête XQuery, on peut construire un document XML. Pour créer les nœuds d'un document XML ; XQuery procède de la manière suivante (figure 5.7) ***[CK 03]***:

- Pour les éléments *Elément, Attributs et Texte, commentaire et les instructions de traitement ,* XQuery utilise la même syntaxe XML.
- Pour le nœud *document*, qui n'a pas de syntaxe explicite en XML, XQuery fournit un constructeur *document {}*. Celui ci crée un nœud document vide.

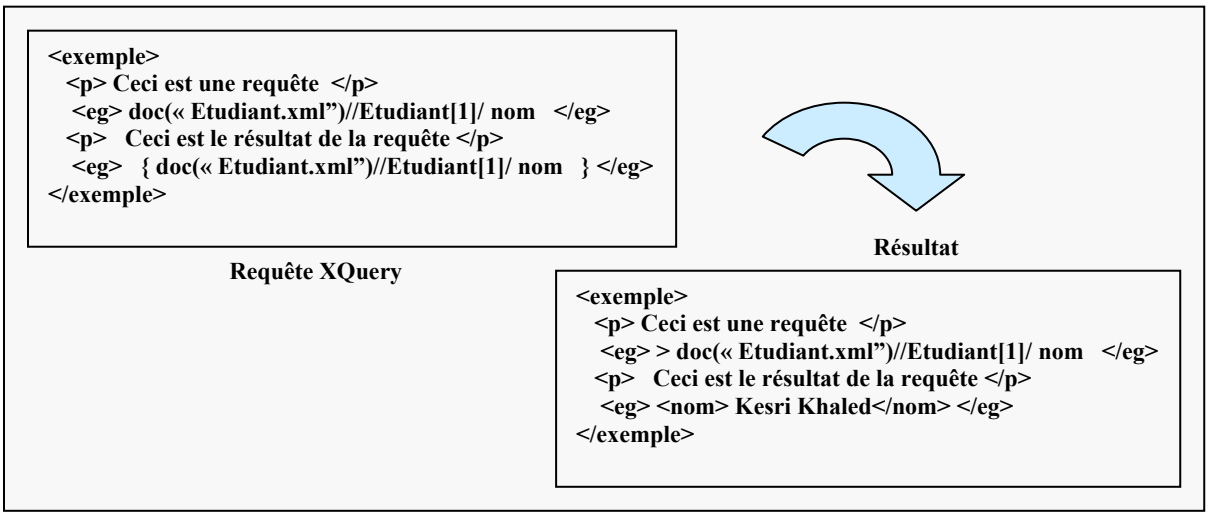

FIG. 5.7 – XQuery : Création de nœud.

7.3 Les expressions FLWOR

Les expressions FLWR : *For, Let, Where* , *Return* et *Order by,* sont l'une des plus puissantes, et des plus utilisées des expressions de XQuery. elles sont similaires aux bloc SELECT-FROM-WHERE-GROUP BY de SQL. Beaucoup plus expressives cependant.***[ROB 04]*** La forme générale d'une expressions FLWR est la suivante:

1. ***for $*** *var* ***in*** *foret [, $ var* ***in*** *foret]**
2. ***let*** *$ var := sous-arbre*
3. ***where*** *condition*
4. ***Order By*** *$var*
5. ***return*** *résultat*

L' exemple (figure 5.8) illustre une requête XQuery sous forme d'expression FLWR .

```
for $b in doc(''Etudiant.xml'')// Etudiant
where $b/@Annee= ''2000''
return $b/ Nom
```

FIG. 5.8 – Expression FLWR

La clause FOR

Permet d'associer une ou plusieurs variables aux expressions, en créant un flux de tuples (*Tuple stream*) ;où chaque tuple lie une variable *$var* donnée, avec un des éléments évalué par l'expression dont la variable *$var* est associée.***[ROB 04]*** Examinons l'exemple suivant qui permet d'obtenir la liste des personnes inscrites dans l'université.

```
FOR $Etud IN //Etudiant
RETURN
$Etud/nom
```

La clause For permet d'effectuer des itérations, sur un ensemble de nœuds obtenus dans une requête. Ici, //Etudiant renvoie la liste des étudiants de l'université. A chaque itération, un élément de cette liste est affecté à la variable $Etud, et le traitement explicité dans le For (ici Return $Etud/nom) est appliqué à cette variable.

La clause LET

Cette clause associe une variable au résultat entier d'une expression. Cette expression peut être absolue ou définie par rapport aux variables d'itération de la clause *For*. ***[ROB 04]***. Si on reprend notre exemple , une deuxième façon d'exprimer la requête précédente est :

```
LET $noms := document("Etudiant.xml")//nom
RETURN $noms
```

La clause LET permet d'affecter des variables (ici $noms) ***[CK 03]***, et d'utiliser ces variables dans la suite de la requête. La requête précédente est donc simple à comprendre : le résultat de la requête //nom est affecté à la variable $noms, et le contenu de cette variable est renvoyé à l'utilisateur.

La clause Where

Cette clause filtre les tuples, en retenant seulement ceux qui satisfassent la condition. La condition est une expression logique de prédicats élémentaires***[CK 03]***. Il est donc possible de filtrer les nœuds, obtenus par la requête dans une clause *For*, à la manière du Where SQL. Par exemple, si on désire obtenir l'établissement de l'étudiant *Kesri Khaled*, une des manières de procéder est la suivante :

```
FOR $Etud IN document("Etudiant.xml") //Etudiant
WHERE $Etud/nom = "Kesri Khaled"
RETURN $Etud//Etablissement
```

Permet de construire les instances des documents résultats, à partir des tuples des variables associées. La clause *Return* est évaluée seulement pour les tuples qui ont survécu à la clause *Where* . Penons pour exemple la requête suivant :

```
FOR $i IN document("Etudiant.xml")// nom
RETURN $i
```

Elle donne le résultat suivant :

```
<nom> Kesri</ nom >
< nom >Hammimed</ nom >
< nom >Abdi</ nom >
```

A chaque itération, le contenu de la variable $i est concaténé à une variable de sortie, qui est renvoyée à l'utilisateur lors de la dernière itération. Il est possible aussi de construire le format XML dans la clause *Return.*

```
FOR  $i  IN  document("Etudiant.xml")//nom
RETURN <nom de famille> $i</ nom de famille >
```

Ce qui donnera le résultat suivant :

```
<nom de famille> Kesri</ nom de famille >
< nom de famille >Hammimed</ nom de famille >
< nom de famille >Abdi</ nom de famille >
```

La clause order by

Cette clause trie les tuples avant que la clause *return* soit évaluée ; afin de changer l'ordre des tuples résultats***[ROB 04].*** Par exemple, la requête suivante liste les noms des étudiants dans l'ordre alphabétique.

```
For $ t in doc(''Etudiant.xml'') // nom
Order by $t
Return $t
```

Le résultat de cette requête est :

```
<nom>Abdi </nom>
<nom>Hammimed </nom>
<nom>Kesri </nom>
```

On peut spécifier également, dans une clause *order by*, l'ordre de trie (ascendant ou descendant), en utilisant des spécifications d'ordre appelées *orderspec. **[ROB 04]***

7.4 Elimination de duplicata

Les données souvent comportent des redondances. En XQuery, les expressions *FLWOR* sont combinées avec la fonction *distinct-values()* pour supprimer les doublons des sous arbres. Cette fonction est illustrée par l'exemple qui suit : ***[ROB 04]***
La requête suivante retourne l'établissement ou chaque étudiant est inscrit.

```
Doc(''Etudiant.xml'')//Inscription/Etablissement
```

Ce qui donne le résultat suivant , sachant qu'un étudiant peut avoir plusieurs inscriptions :

```
< Etablissement >USTHB</ Etablissement >
< Etablissement>BLIDA</ Etablissement >
< Etablissement>USTHB</ Etablissement >
<Etablissement>USTHB</ Etablissement >
```

En utilisant la fonction *distinct-values ()* , on obtient la requête suivante :

```
distinct-values(Doc(''Etudiant.xml'') //Inscription/Etablissement )
```

et on obtient le résultat suivant :

```
< Etablissement >USTHB</ Etablissement >
< Etablissement>BLIDA</ Etablissement >
```

7.5 Les jointures

Une requête peut associer plusieurs variables dans une clause *For,* afin de combiner des informations à partir de plusieurs expressions, et éventuellement à partir de plusieurs sources de données. Un résultat comportant toutes les combinaisons possibles, des éléments des sources impliquées représente *un produit cartésien.* Si on applique sur celui ci une clause *where*, on obtient alors ce qu'on appelle communément une *jointure* :***[ROB 04][ROB 03]***

Les opérations de jointures sont particulièrement intéressantes pour notre étude, puisqu'elles permettent de définir des vues sur les données XML. Par exemple , supposons avoir un fichier appelé « *bourse.xml »* qui continent les informations des bourses des étudiants du fichier « etudiants.xml » (figure 5.9) :

```
<Boursier>
    <nom> Kesri </nom>
    <Montant>200 da </ Montant >
    <CodeOeuvre> CUB 3</CodeOeuvre>
</Boursier>
<Boursier>
    <nom> Hammimed</nom>
    < Montant >250 da </ Montant >
    <CodeOeuvre> ALIA </CodeOeuvre>
</Boursier>
```

FIG. 5.9- – Document « Boursier.XML ».

En se référant à nos données sur les étudiants décrites précédemment dans ce chapitre, on voudrait connaître le montant de la bourse ainsi que le nom et le prénom de chaque étudiant. La requête suivante exécute une jointure pour combiner les données des étudiants avec les données de la bourse :

```
For    $etud in doc(''Etudiant.xml'')//Etudiant,
       $Bour in doc(''Boursier.xml'')//Boursier
where $Etud/nom = $Bour/nom
return
<Information bourse>
{ $Etud/nom, $Etud/prénom, $Bour/montant}
</ information bourse>
```

Le résultat de la requête est le suivant :

```
<Information Bourse>
    <nom> Kesri </nom>
    <Prénom> Khaled </Prénom>
    < Montant >200 da </ Montant >
</ Information Bourse >

< Information Bourse >
    <nom> Hammimed</nom>
    <Prénom> Leila </Prénom>
    < Montant >250 da </ Montant >
</ Information Bourse >
```

7.6 Les quantificateurs (*quantifiant*)

Certaines requêtes ont besoin de déterminer si au moins un des éléments d'une séquence satisfasse une condition ;Ou par contre, si tous les éléments d'une séquence satisfassent cette condition. XQuery permet d'exprimer ces notions en utilisant *les quantificateurs existentiel, some-satisfies* et *every-satisfies* . :***[ROB 04]***

Dans la requête suivante, Le quantificateur *Some* teste s'il existe au moins un étudiant inscrit à l'université de Blida.

```
For $b in doc (''Etudiant.xml'')//Etudiant
Where some $a in $b/Inscription
        Satisfies  ( $a/Etablissement = "UBLIDA")
Return $b/nom
```

De la même manière, dans la requête suivante , on utilise le quantificateur universel *every-staisfies* pour tester si tout les étudiants sont inscrits à l'université de blida.

```
For $b in doc (''Etudiant.xml'')//Etudiant
Where Every $a in $b/Inscription
        Satisfies  ( $a/Etablissement = "UBLIDA")
Return $b/nom
```

7.7 Les expressions conditionnelles

XQuery permet d'utiliser des expressions conditionnelles dans les requêtes:***[ROB 04].*** L'exemple suivant illustre l'utilisation de *if-then-else* pour lister les noms des étudiants qui ont plus qu'une inscription.

```
For $Etud in doc(« Etudiant.xml »)//Etudiant
Return
  If  (count($Etud / Inscription )>  0  )
   Then  $Etud / nom
   Else ()
```

Le résultat de la requête est :

```
<nom>  Kesri </nom>
```

8. Architectures de médiation existantes

8.1 GAV versus LAV

La plupart des architectures de médiation se sont orientées vers l'architecture DARPA I3. ***[WG 93] [DAN 03a].*** On peut classifier ces systèmes suivant la relation entre les schémas des sources locales par rapport au schéma unifié global sur le médiateur. On distingue deux approches : une vue globale comme vue sur des vues locales *GAV (Global As View)* ou des vues locales comme vues du schéma global *LAV (Local As View)(* Figure 5.10) ***[DAN 03a].***

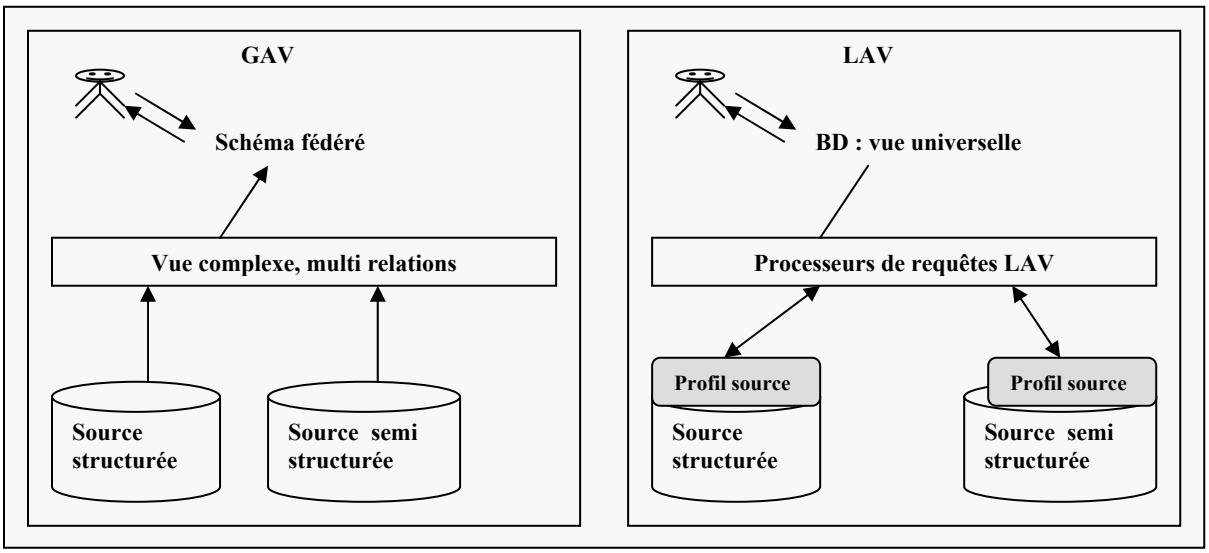

FIG. 5.10 – Architectures GAV et LAV

Dans l'approche *GAV* , la transformation d'une requête sur le schéma global en requête sur le schéma local est une opération de composition de vues simple. Dans une approche *LAV,* la requête sur le schéma global doit être reformulée suivant les schémas des sources locales. Dans une architecture GAV, une modification sur l'ensemble des sources locales ou sur leurs schémas entraîne une reconsidération du schéma global ; par contre dans l'approche *LAV* où chaque source est spécifiée de manière indépendante , un changement local de schéma entraîne la mise à jour de la vue locale.

GAV est l'approche la plus utilisée (TSIMMIS, XPeranto, e-xmlMedia) ; très peu utilise l'approche *LAV* (AGORA). Dans la suite, nous détaillons ces différentes architectures de médiation.

8.2 TSIMMIS

TSSIMIS (*the Stanford-IBM Manager of Multiple Information Sources)* ***[CHA 94]*** est un projet permettant de développer des outils qui facilitent l'intégration, de sources d'informations hétérogènes comportant des données structurées et semi structurées ***[MOL 95]***

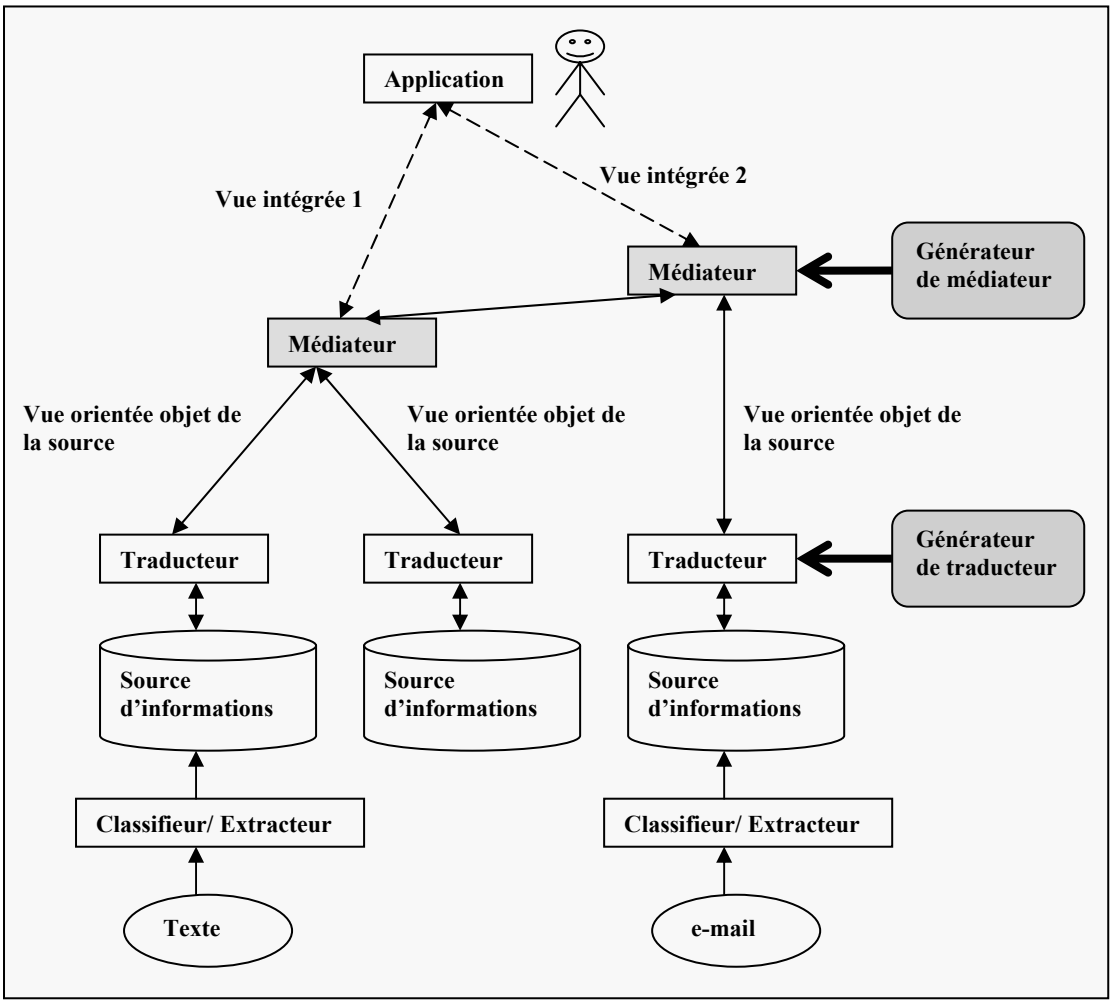

FIG. 5. 11 – Architecture de TSIMMIS

TSIMMIS adopte l'architecture médiateurs/adaptateurs, il utilise le modèle OEM comme modèle commun , un langage de requête OEM-QL ***[MOL 97]*** a été développé afin de traiter des requêtes sur OEM. TSIMMIS a une architecture composée de (Figure 5.11) : ***[MOL 95] [CHA 94]***

- *Le classifieur / extracteur :* permet de classifier automatiquement les informations non structurées, et d'en extraire des attributs clés. Ainsi , il peut déterminer qu'un fichier est un message e-mail, et d'en extraire l'auteur, la date et le contenu. Ce composant est basé sur le système RUFUS développé au centre de recherche d'IBM Almaden.

- *Le traducteur :* permet la transformation des requêtes et des données. Il convertit les requêtes OEM-QL sur le modèle OEM ,en requêtes spécifiques à la source. Il convertit ensuite les données résultats de la source, en données du modèle commun OEM.

- *Le médiateur :* lors de la réception d'une requête globale, le médiateur permet de spécifier les sources adéquates de chaque sous requête. Et lors de l'envoie du résultat à l'utilisateur, il permet de combiner les informations des différentes sources.

- *Les générateurs de médiateurs & traducteurs :* ils servent à la création automatique ou semi automatique de médiateurs (resp. traducteurs)**.** A partir d'une description de haut niveau des informations nécessaires pour le traitement, ils procèdent au génération de ces composants.

- *L'interface client :* médiateurs et traducteurs , reçoivent tout deux des requêtes OEM-QL en entrée, et retournent des objets OEM. L'utilisateur, peut accéder aux informations, soit en développant des applications qui traitent des objets OEM ;ou en utilisant une application MOBI *(MOsaic Based Information Explorer)* ,qui permet de se connecter aux médiateurs et/ou traducteurs, et de spécifier des requêtes OEM-QL dans une interface web.

8.3 STRUDEL

SRTUDEL est un système qui applique les concepts de gestion des bases de données, au gestion de contenu web. En particulier, il permet de définir des spécifications déclaratives du contenu, et de la structure des sites web. A partir de ces spécifications, STRUDEL permet de générer automatiquement des sites web navigables.***[FRN 98a][FRN 98b]***L'idée principale est de séparer la gestion des données d'un site ; la création et la gestion de la structure du site ; de la présentation graphique des pages webs.

STRUDEL crée en premier, une vue intégrée (modèle uniforme) des données qui seront mises sur le site. Les données résident soit, dans un entrepôt de données (*repository*) interne à *STRUDEL* ,soit dans des sources externes hétérogènes (bases de données, textes…etc.). Les données, internes ou externes dans STRUDEL, sont modélisées par des graphes orientés étiquetés utilisés pour les données semi structurées, similaires à OEM ***[FRN 98a][FRN 98b]***. Un ensemble d'adaptateurs spécifiques convertit la représentation externe en graphe. La vue intégrée des données est appelée *graphe de données.*

Deuxièmement, le constructeur de site spécifie la structure du site, en utilisant une *requête de définition de site,* écrite dans le langage de STRUDEL appelé *StruQl.* Le résultat de l'évaluation de la requête de définition de site, sur le graphe de données est appelé *graphe de site*. Ce graphe modélise le contenu et la structure du site.

Troisièmement le constructeur spécifie la présentation visuelle des pages dans un langage de modèle (*template*) de pages HTML. Le générateur HTML produit le texte HTML pour chaque nœud dans le graphe du site à partir d'un modèle HTML correspondant. Le résultat est un site web HTML navigable.***[FRN 98a] [FRN 98b]***

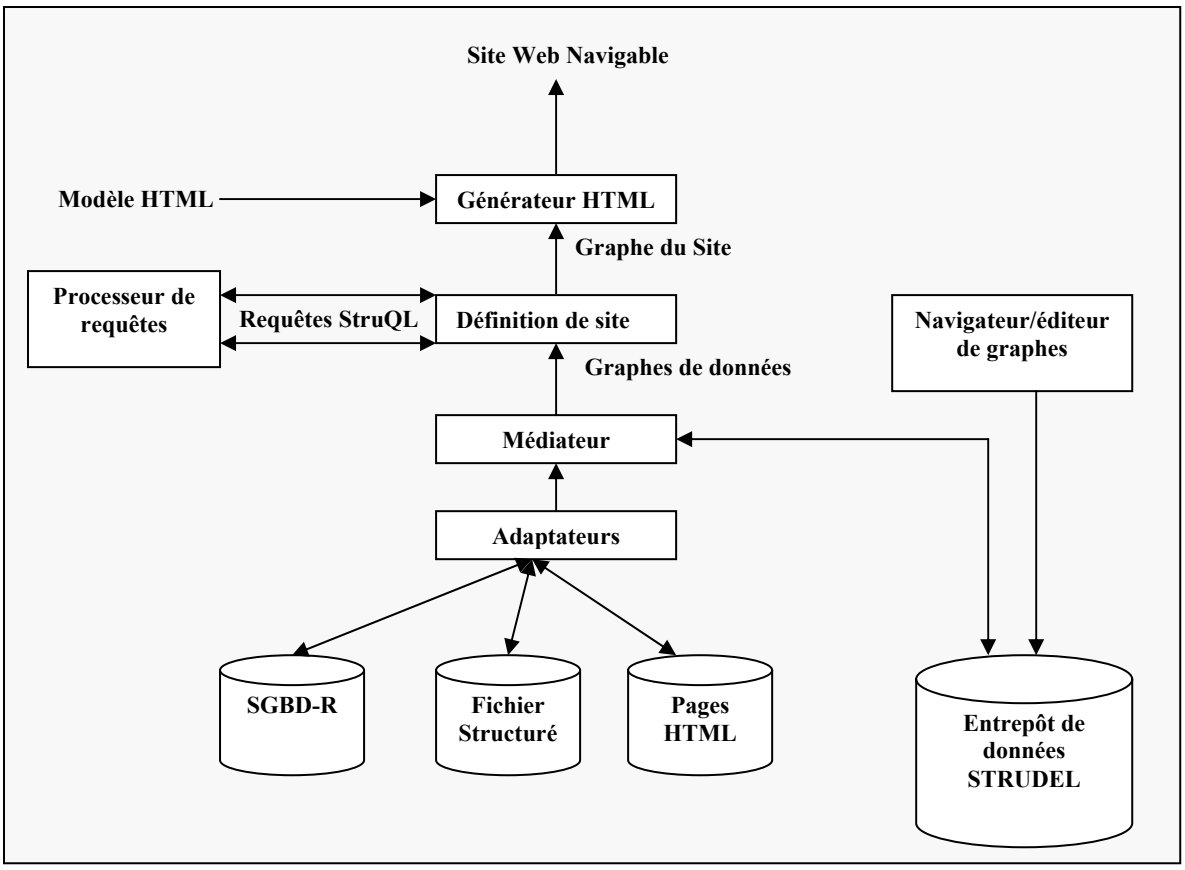

FIG. 5.12 - Architecture de STRUDEL

La figure précédente 5.12, décrit le processus d'exécution de Strudel, de la perspective du *constructeur du site.* Il est composé de : ***[FRN 98a] [FRN 98b]***

- *Un entrepôt de données :* le *graphe de données* et le *graphe de site* sont stockés dans l'entrepôt de données *STRUDEL.* Les données sont obtenues des adaptateurs qui convertissent les données externes dans un format interne.

- *Le médiateur :* fournit une vue uniforme de toutes les données sous jacentes, indépendamment de son lieu de stockage initial. Pour cela il utilise l'approche d'intégration dite *Warhouse,* où le résultat de l'intégration des données est stocké dans l'entrepôt de données *Strudel* .

- *Processeur de requêtes : STRUDEL* définit le langage *STRUQL* pour exécuter des requêtes, et restructurer des données semi structurées. L'interpréteur de requêtes de *STRUDEL* utilise les opérateurs traditionnels (jointure, restriction et sélection) pour transformer une requête *STRUQL* en un arbre algébrique, pour optimisation.**[FRN 98a]**

- *Générateur HTML :* il permet de produire la représentation graphique de chaque page du site Web. Un modèle HTML est associé à chaque nœud du graphe du site. Le résultat est un site Web navigable.

8.4 AGORA / LeSelect

AGORA est un système d'intégration de données hétérogènes, qui utilise XML comme format d'interface utilisateur, alors que les flux de données dans le processeur de requêtes consistent dans des tuples relationnels. Les requêtes sont écrites en utilisant un langage de requête XML, et les résultats sont formatés en documents XML ; en rendant transparent les traitements relationnels sous jacents. ***[MAN 01]***

Agora est implanté au dessus du système d'intégration de donnée *LeSelect,* développé dans le projet *Caravel* (INRIA Recqencourt) ***[MAN 00]***. *LeSelect* est un système d'intégration, qui utilise le modèle relationnel comme modèle commun des données. Le but *d'AGORA* est de compléter les fonctionnalités de *LeSelect*, en définissant comment : ***[MAN 00]***

- Définir un schéma *d'intégration virtuel, générique et relationnel ;* qui décrit le contenu des documents XML.

- Traduire des requêtes spécifiques à XML, dans le schéma d'intégration relationnel. Et réécrire les requêtes résultats, en utilisant les définitions de vues qui décrivent les documents XML.

- Etendre les mécanismes d'optimisations de requêtes, afin de pouvoir traiter les problèmes liés aux chemins ; lors de la génération du plan d'exécution à travers les sources hétérogènes .

- Indexer le texte, afin d'améliorer les performances de recherche.

Le traitement d'une requête dans le système *Agora/LeSelect* suit le scénario suivant : ***[MAN 01] [MAN 00]*** (figure 5.13)

1. *Formulation d'une requête XQuery :* ceci se passe des deux manières suivantes :

   - Les utilisateurs experts en structures de données, peuvent écrire des requêtes XQuery complètes .
   - Les utilisateurs novices peuvent utiliser une GUI (*interface graphique utilisateur*), qui permet la navigation dans les données. Les requêtes seront raffinées progressivement grâce à celle ci.

2. *Traduction de la requête XQuery dans SQL :* la requête XQuery ,est ensuite traduite dans un ensemble de requêtes SQL sur le schéma relationnel générique. Ces requêtes sont équivalentes avec la requête XQuery originale .

3. *Réécriture des requêtes SQL en utilisant les vues* : la requête SQL obtenue de la phase précédente, est alors réécrite en une requête équivalente qui utilise seulement les vues, qui modélisent les chemins d'accès réels aux données XML natives.

4. *L'assemblage du résultat en XML* : le résultat de l'exécution est un ensemble de tuples qui sont organisés dans des documents XML finaux ; afin d'être présentés à l'utilisateur.

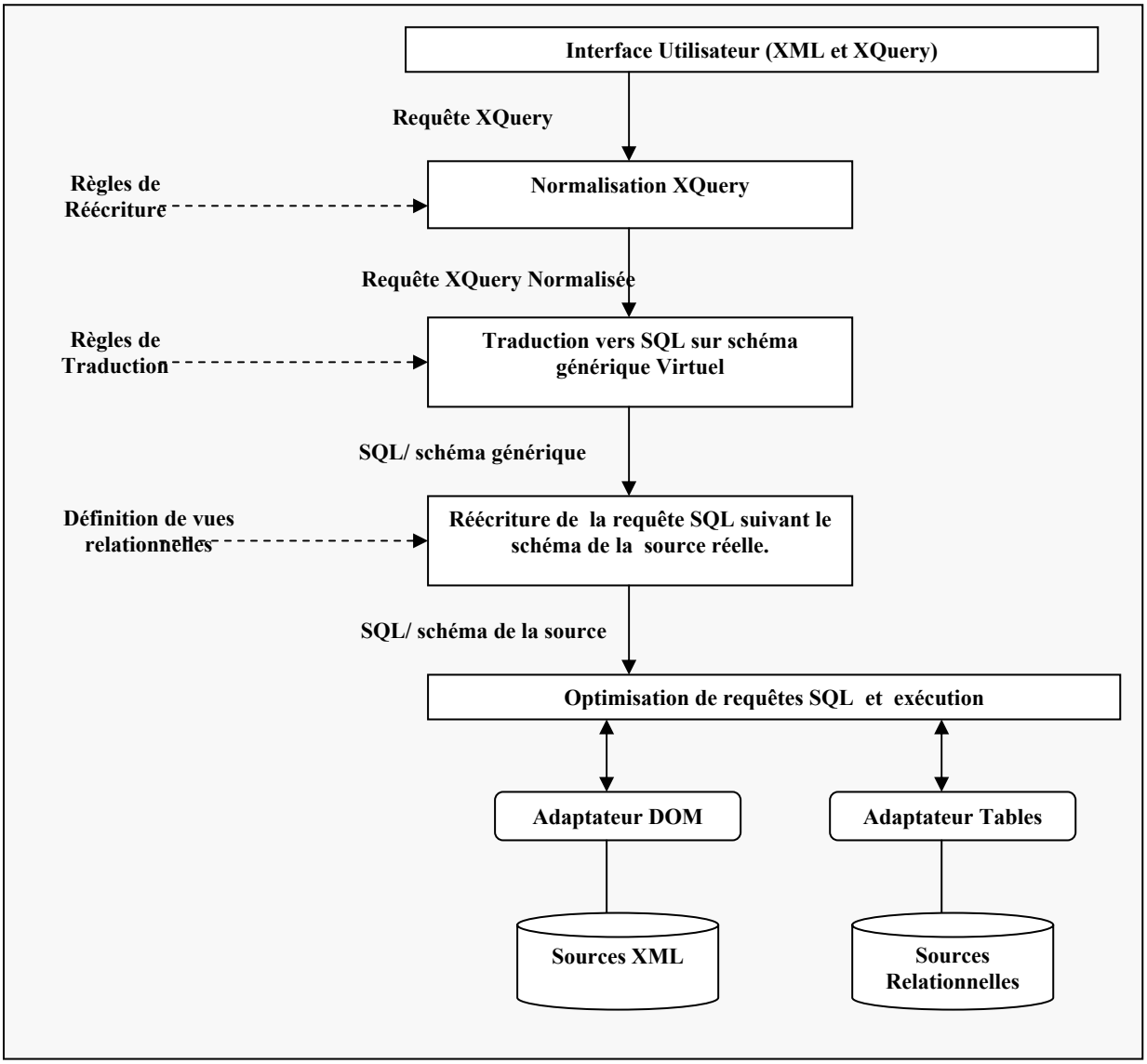

FIG. 5.13 - Architecture du système AGORA

8.5 XPeranto

XPeranto ***[SHA 01]*** (XML *Publishing of Entities, Relationships, ANd Typed Object*) est le nom de code de la suite d'intégration de données d'IBM. XPeranto permet donc, de publier de manière flexible et générale , des données relationnelles en XML en créant des vues XML sur ces données. XPeranto est composé d'un processeur de requêtes XQuery, d'un système de médiation permettant de récupérer des données hétérogènes ; de diverses sources de données (SGBD relationnel, document texte…etc.) ; et un composant permettant de présenter les résultats de telle façon , qu'ils puissent être intégrés dans des applications clientes.

De manière générale, XPeranto utilise deux mécanismes pour fédérer et intégrer des données relationnelles : ***[SHA 01]***

- Un mécanisme dit *calcul Push down* d'évaluation de requêtes XQuery. Il permet de fragmenter la requête globale en sous requêtes ; ensuite de faire descendre les calculs

intensifs, dus aux procédures d'évaluation de ces sous requêtes, au niveau du moteur relationnel sous jacent.

- Des mécanismes de compositions de vues, permettent d'éliminer les fragments XML intermédiaires qui n'apparaissent pas dans le résultat final.

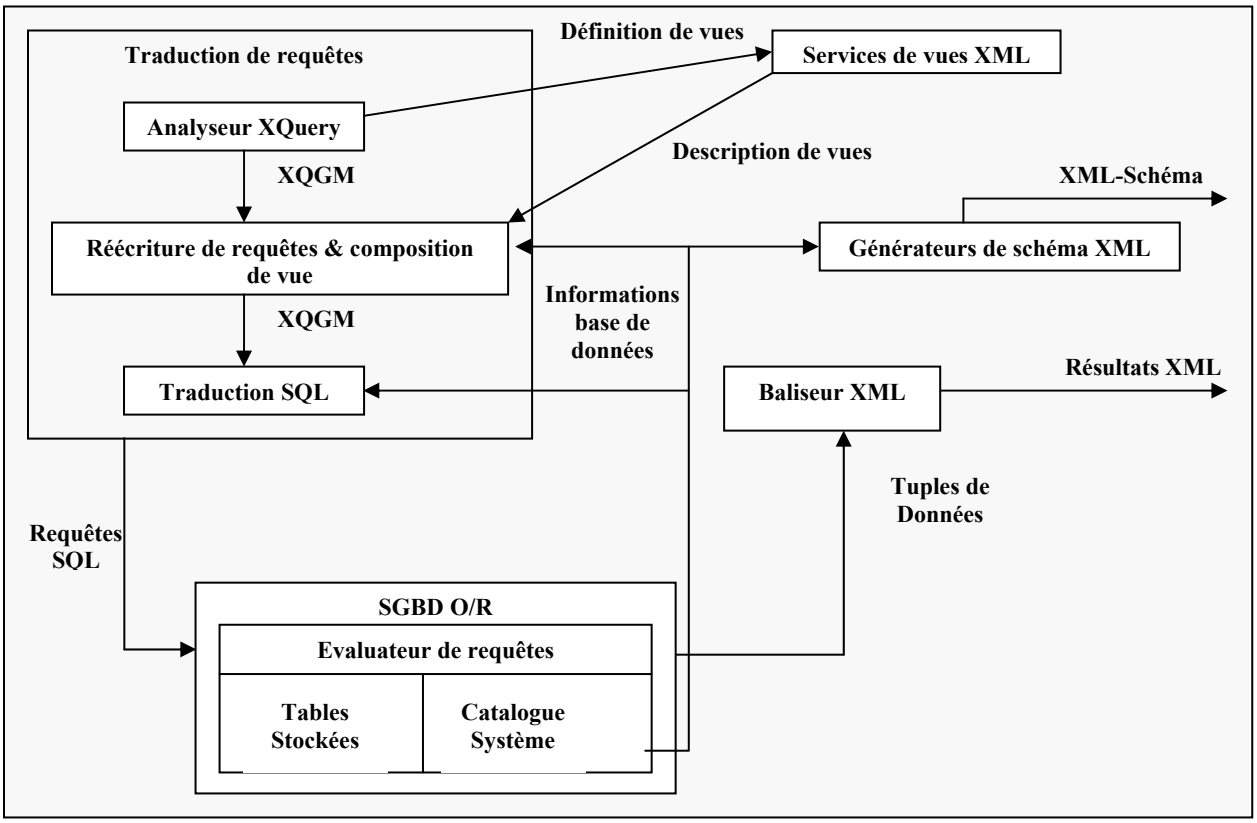

FIG. 5.14 - Architecture de XPeranto

La figure 5.14. décrit l'architecture de XPeranto , ses principaux composants sont :

- *Le traducteur de requête :* il traduit la requête XQuery en une requête, écrite dans le langage natif des SGBDs *objets* ou *relationnels* sous jacents. Le composant *analyseur XQuery* traduit la requête XQuery dans une représentation interne appelée *XQGM (XML Query Graph Model)*. Le composant *réécriture de requête& composition de vues,* prend la représentation XQGM de la requête , résout les références sur les vues, compose la vue XML ,et construit une représentation XQGM équivalente. Le composant de *traduction SQL* traduit XQGM en SQL.

- *Un service de vues XML :* c'est une interface de stockage et de chargement des données, pour les définitions de vues XQuery.

- *Un générateur de schéma XML :* à partir des données relationnelles, ce composant produit des informations pour générer des vues XML.
- *Un baliseur XML :* il construit la requête XML résultat ,à partir des résultats tabulaires SQL.

8.6 La suite d'intégration *e-XMLMedia*

La suite e-XMLMedia est une collection de composants d'intégration de données hétérogènes. Ils implémentent une architecture de médiation entre différentes sources hétérogènes, développée durant le projet MIROWEB (Laboratoire PRISM) ***[GAR 99a] [DAN 03b] [DAN 03a].*** Le modèle de données semi structurées à été choisi comme modèle pivot.

A l'instar des autres médiateurs cités dans ce chapitre, la suite e-XMLMedia adopte l'architecture I3 de DARP à bases de médiateurs et adaptateurs ***[DAN 03a][GAR 02a] [GAR 04][Gar & al 03b].*** La spécificité de l'architecture décrite dans ***[DAN 03a]*** ,est que celle ci est entièrement conçue pour gérer des données semi structurées, et plus particulièrement XML. La figure 5.15. décrit l'architecture de médiateur / adaptateurs utilisée. Le cœur de l'architecture est le médiateur lui même. Ce médiateur est décomposé en plusieurs modules : ***[DAN 03b] [GAR 02c]***

- *L'analyseur :* il décompose la requête de l'utilisateur en une structure interne, susceptible d'être manipulée facilement par les différents composants du médiateur. Il vérifie aussi si la requête est *valide* , aussi bien *syntaxiquement* ,que par rapport aux types de données interrogées.

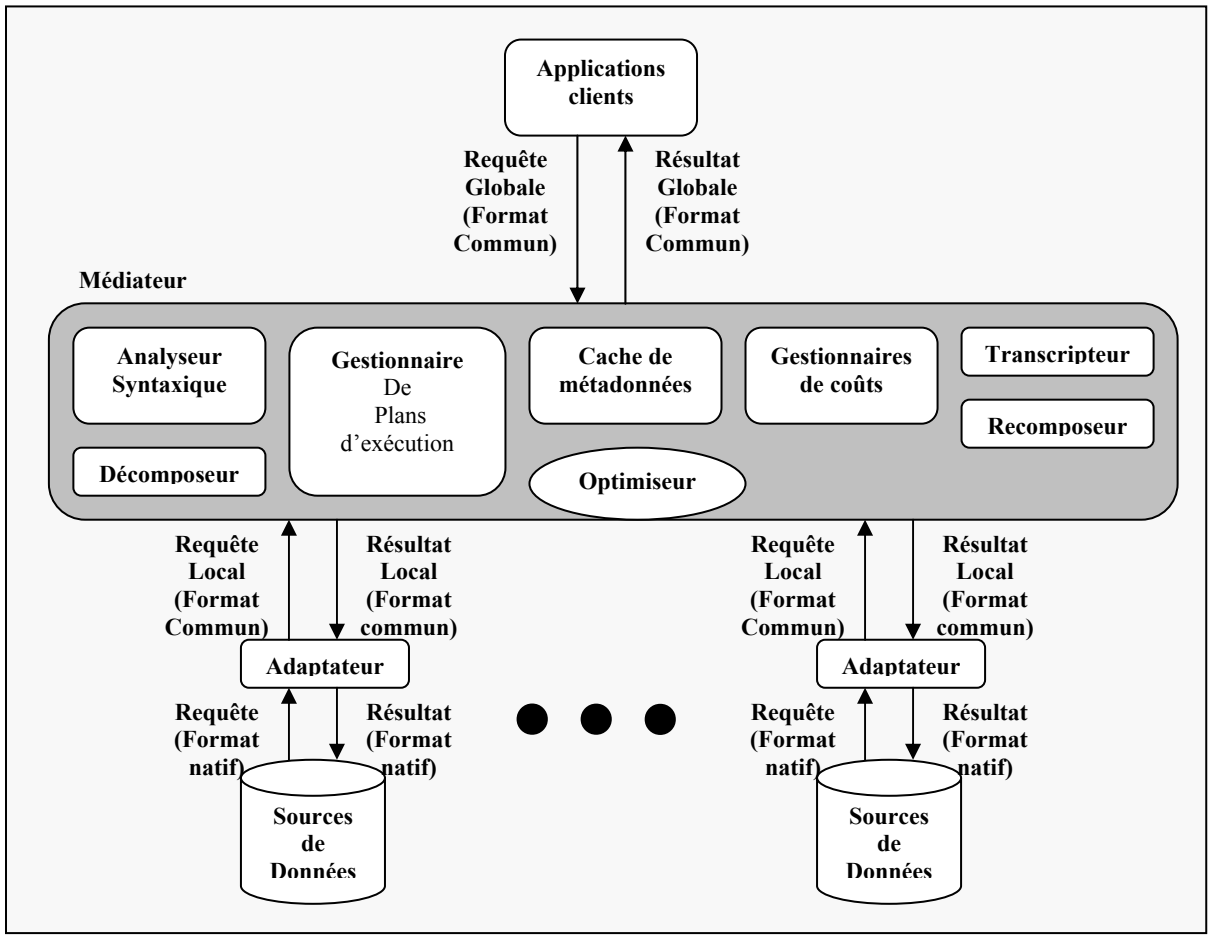

FIG. 5.15 – Architecture générale du médiateur e-xmlMedia

- *Le décomposeur :* son rôle est d'analyser la structure de la requête, afin de déterminer comment décomposer la requête initiale en sous requêtes.

- *Le cache des métadonnées :* il se charge de conserver au fur et à mesure, les localisations des données et les schémas des différentes données réparties dans les sources.

- *Le gestionnaire des plans d'exécutions :* il permet de générer l'ensemble des plans d'exécution possibles pour satisfaire une requête donnée.

- *Le gestionnaire de coût :* il estime le coût d'exécution du plan.

- *L'optimiseur :* il détermine en fonction des composants ci dessus, quel est le plan optimal à choisir pour évaluer la requête.

- *Le recomposeur :* il restructure et recompose les résultats donnés par les différents adaptateurs.

- *Le transcripteur :* il transforme la structure interne du résultat, en un format lisible par l'utilisateur (XML).

La société *e-XMLMedia* à implanter cette architecture à travers deux composants essentiels ***[GAR 02c]***: (figure 5.16)

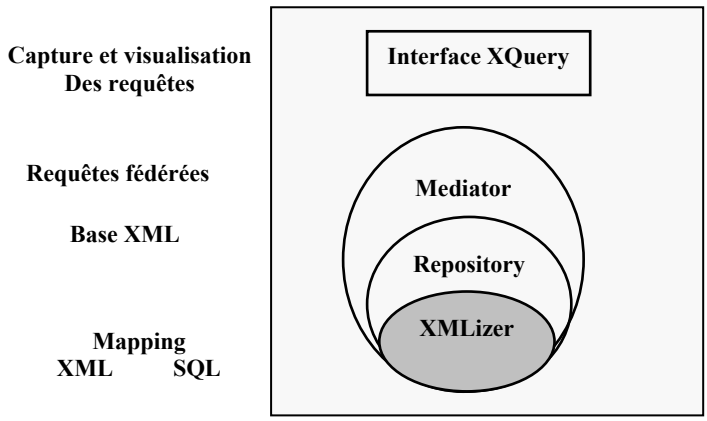

FIG. 5.16 - Les composants de l'intégration de données e-XMLMedia

- *L'entrepôt e-XML Repository :* ce composant permet de stocker des documents XML dans des SGBDs objet ou relationnels quelconques. Il fournit via XQuery, le moyen d'extraire des fragments de documents XML et de les assembler. Il se compose des modules suivants : ***[GAR 02a][Gar & al 03b]***

  1. *Module de stockage :*il permet de traduire (*mapping)* des documents XML, dans une table relationnelle. Quand les données XML n'ont pas de schéma, l'entrepôt e-XML génère les méta données sous forme de guide de données.

  2. *Le module de requêtes :* le langage de requête utilisé est XQuery , mais seulement les opérations d'insertions et de suppressions sont supportées. Les requêtes peuvent

exécuter des jointures sur plusieurs collections XML. Les résultats de ces requêtes sont des fragments de documents XML.

- *Le médiateur e-XMLMedia* ***[DAN 03b]***:Le médiateur fournit une vue uniforme de plusieurs sources de données hétérogènes, en utilisant le langage XQuery. Il est composé des modules suivants :

  - *Une console d'administration :* pour administrer le médiateur, et enregistrer les adaptateurs des sources de données.

  - *L'évaluateur :* est responsable du traitement des requêtes : l'analyse, la décomposition, l'optimisation, et la composition de résultats.

  - *L'exchanger :* fournit l'interface de communication, entre les médiateurs et les adaptateurs ; il gère aussi , le cache des métadonnées.

  - *Un adaptateur :* est capable d'exécuter une requête envoyée par le médiateur sur une source de données spécifique. Puis de fournir le résultat sous forme d'arbre d'objets DOM.

9. Synthèse

Les systèmes présentés précédemment adoptent tous l'architecture DARPA I3 ***[WG 93]***qui repose sur un ensemble de médiateur/adaptateurs. Apres le modèle OEM (TSIMMIS), le modèle de données XML fut utilisé par la majorité de ces systèmes. AGORA a choisi une approche relationnelle en se basant sur la traduction d'une requête portant sur des données semi structurées vers une requête relationnelle.

Le premier langage de requête pour données semi structurées à été LOREL (TSIMMIS),puis XML-QL,QUILT et XQuery (AGORA). Celui ci standardisé depuis peu, semble être le plus approprié pour la manipulation des données semi structurées.

Dans une architecture de fédération de données basée sur le modèle médiateur/adaptateurs, on peut utiliser un schéma global comme vue sur des schémas locaux (GAV) ou des vues locales sur schéma global (LAV). Seul AGORA utilise l'approche LAV.

9.Conclusion

Nous avons présenté dans ce chapitre une étude sur les systèmes de médiation s'appuyant sur le modèle XML comme modèle pivot ; ainsi que les langages de requêtes, les plus importants.

XQuery est un langage de requête type SQL. Il est puissant, et permet d'exprimer des requêtes très complexes, à savoir des expressions de chemins, des jointures , des expressions conditionnelles…ext***[BRA 03a]***. De ce fait, il est incontestablement, le langage de requête le plus adéquat pour données XML.

La plupart des architectures de médiation , vues dans ce chapitre utilise le modèle de données XML, soit comme modèle de données, soit comme interface utilisateur. Ils ont tous, adopté l'architecture de médiateur /adaptateurs ; Où le médiateur s'occupe de la distribution des données ; alors que l'adaptateur s'occupe de l'hétérogénéité de celles ci.

Dans ce contexte, l'évaluation d'une requête procède de deux façons : Exporter toutes les données en XML, et exécuter les requêtes par un moteur XML ***[GAR 03b] [FAN 02]*** ; ou traduire les requêtes dans le modèle de la source sous jacente, afin qu'elles puissent être exécutées par le moteur de celles ci ***[GRU 04] [SHA 01] [MAN 00] [MAN 01]***.

Par conséquent , lors de l'élaboration d'un système de bases de données fédérées à travers une architecture de médiation à base de XML , il faut tenir compte des éléments suivants :

- Le choix du langage de requête.
- Le mode d'évaluation des requêtes.
- Conception d'une architecture adéquate aux choix cités ci dessus.

Chapitre 6 : Une Architecture de Médiation « Tout-Relationnel »

1. Introduction

Les systèmes de fédération de données doivent être capable de prendre en charge - grâce à une architecture de médiateur /Adaptateurs- différentes sources hétérogènes, disparates sur le réseau. Ce qu'il ne faut pas négliger, est que la plupart des informations disponibles actuellement sont stockées dans des bases relationnelles ; et que le langage SQL est mature, et bien implanté. Il est donc important , lors de l'élaboration de tels systèmes, de non seulement récupérer les applications clients existantes, mais aussi les compétences des équipes de développements déjà expertes dans le modèle relationnel.

Par ailleurs, cette solution s'avère nécessaire, si pour des besoins organisationnels, fonctionnels ou d'extension, l'entreprise se voit obliger d'utiliser un système d'information distribué. Il serait moins coûteux , de continuer à avoir l'illusion d'un seul système homogène centralisé, en utilisant les mêmes interfaces utilisateurs.

Dans ce chapitre nous présentons l'architecture de médiation que nous préconisons. Elle est basée sur le modèle XML et présente l'avantage d'utiliser des applications clientes existantes, puisqu'elle prévoit, à la différence des autres architectures, un modèle d'intégration de vues, capable de prendre en charge des traductions du langage SQL vers le langage commun XQuery. En effet, Les requêtes sont posées en utilisant le langage SQL et les résultats sont formatées sous formes de tuples relationnels. Notre but étant d'étudier la faisabilité d'un système d'intégration basée sur XML avec l'illusion « tout-Relationnel ». Plus particulièrement on décrit comment :

- Définir un schéma d'intégration virtuel qui décrit le contenu de nos différentes bases de données.
- Traduire la requête SQL écrite sur une schéma relationnel vers une requête XQuery écrite sur le schéma d'intégration et comment réécrire la réponse de la requête en utilisant une définition de vue.
- Exportation des données SQL en XML ,et comment utiliser les ressources du moteur SQL sous jacents pour exécuter les requêtes XQuery globales.

Dans la section (2) nous présentons notre approche de médiation . on argumente dans la section (3) notre choix du modèle XML ,comme modèle commun. Dans la section (4), on présente l'architecture des schémas sur laquelle est basé notre approche. La technique d'exportation de données relationnelles en XML est expliquée en (5). La section (6) présente le médiateur et ses principaux fonctions. Une architecture fonctionnelle pour le système est proposée en (7). La section (8) décrit un exemple d'utilisation de notre approche. On conclut en (9).

2. Une architecture de système médiation

Notre approche s'inspire des différentes architectures citées dans le chapitre précédent. Elle est basée sur une couche de médiateur / adaptateurs ; avec la particularité de permettre l'intégration d'interfaces clientes pré-existantes, dans le système de fédération ; sans impact ou avec impact minimum. Le système permet d'intégrer des sources relationnelles hétérogènes, en utilisant le modèle XML \ XQuery comme modèle commun. L'idée de base est de donner à l'utilisateur final, l'illusion d'un système « Tout-Relationnel » ; en ajoutant une couche supplémentaire au niveau des *médiateurs,* permettant la *correspondance*

(*mapping)* entre les requêtes utilisateurs SQL et les requêtes du modèle commun XQuery. Cette approche , permet d'une part d'exploiter les investissements de l'entreprise, et d'une autre part de préserver l'expertise SQL acquise.

Notre architecture est composée des éléments suivants (figure 6.1):

- *L'adaptateur (Wrapper)* : au dessus de chaque source, se place l'adaptateur qui convertit de manière logique les données relationnelles sous jacentes. Cette traduction consiste dans l'exportation du schéma de la source qui lui est associée en XML, et la traduction de la requête XQuery en requête dans le modèle local, puis de convertir les données retournées par la source dans le modèle commun XML.

- *Les médiateurs :* au dessus des adaptateurs, se trouvent les médiateurs. Le médiateur est un composant complexe qui simplifie, réduit, fusionne et explique les données ***[ABI 00].*** Dans notre approche, le médiateur permet de disposer d'un mécanisme de correspondance entre modèle relationnel et XML ; en définissant un ensemble de vues intégrées capable de donner l'illusion du « tout – relationnel ». Il a entre autres deux fonctionnalités majeures : la traduction des données et l'intégration des données en utilisant des spécifications déclaratives.

- *L'interface utilisateur :* l'interface client , est l'infrastructure capable de communiquer entre les différentes GUIs (graphiques utilisateurs interfaces) et les médiateurs. Elle permet de recevoir les requêtes SQL des utilisateurs et de les transmettre à la couche sous jacente. Et de transmettre les tuples résultats.

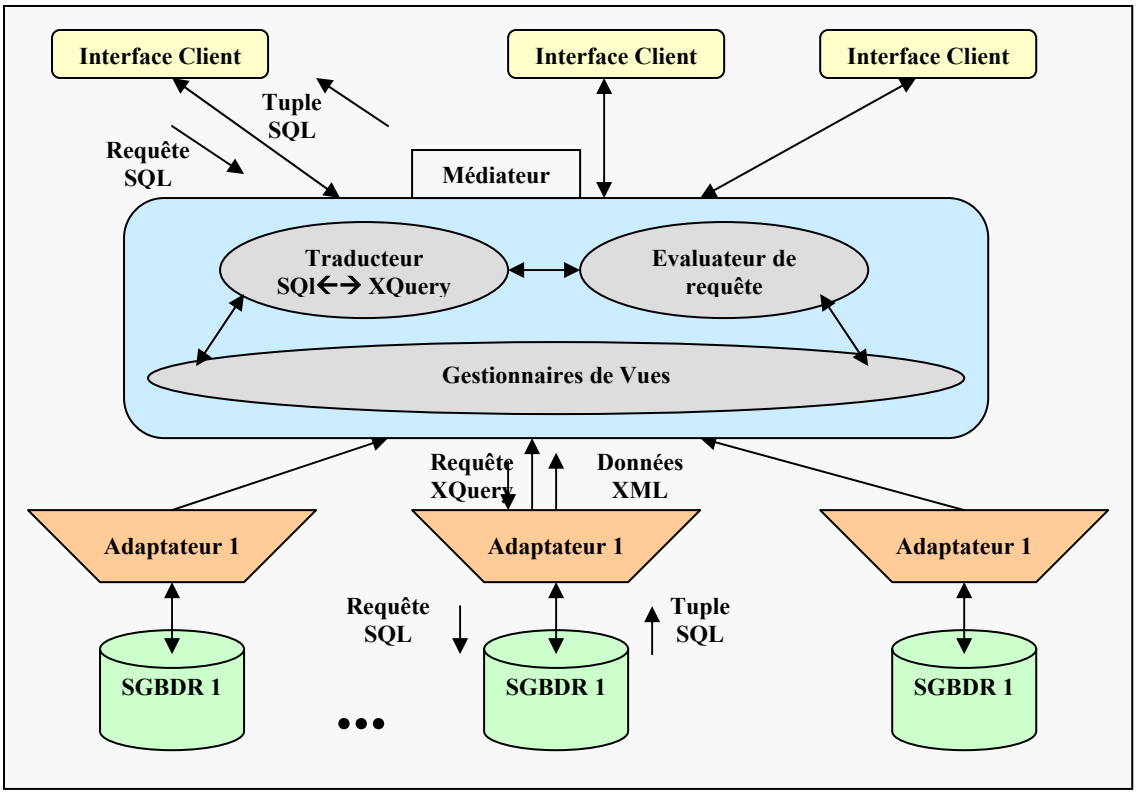

FIG. 6.1 – Architecture de médiation avec intégration de client SQL

Quand une application cliente, lance une requête SQL au système. Celui ci, muni d'un mécanisme de traduction SQL-XQuery ,basé sur un ensemble de vues intégrées, permet de traduire cette requête en requête XQuery dans le modèle commun. Cette requête est ensuite décomposée, les différentes sources participants à la requête sont définies. Chaque sous requête XQuery est envoyée à l'adaptateur correspondant. Enfin au niveau de chaque adaptateur, la requête n'est pas exécutée directement mais , transmise après transformation au moteur relationnel sous jacents pour exécution. En effet, on choisit d'exploiter pleinement le moteur de requête SQL sous jacent, puisque les moteurs de requêtes pour les données XML natives sont encore immatures et n'ont pas encore atteint les performances des moteurs SGBDs.

Finalement , Les résultats transmis en tuples, seront traduits en document XML afin d'être renvoyés au médiateur. Au niveau médiateur , les résultats seront composés , retraduits une dernières fois en tuples et remis à l'interface client.

3. Le modèle de données commun

Le modèle relationnel est un modèle de données simple, expressif, et doté d'une base d'algèbre formelle puissante. Il présenta des limites quant à son utilisation comme modèle commun dans une architecture fédérées. Entre autre :

- *Mauvais support de l'imbrication :* le clause GROUP BY est limitée, les requêtes imbriquées sont difficiles à utiliser , et les mécanismes de référencement existant ne permettent pas d'avoir une vison claire des données représentées.

- *Peu adapté à XML :* la représentation tabulaire supportée par le modèle relationnel ne permet pas une manipulation aisée des données semi structurées.

Le modèle objet fut utilisé par la suite ; mais son implémentation qui revenait généralement au relationnel a limité le développement de bases de données fédérées avec ce modèle. Actuellement, la plupart des systèmes se sont orientés vers l'utilisation de XML comme modèle commun.

Notre approche consiste dans l'utilisation du Modèle XML comme modèle d'intégration de données. Car il présente les avantages suivants : ***[BRA 03b] [DAN 03a]*** abondance des descriptions et typages des données , clarté et extensibilité. Il permet de traduire facilement les modèles de données existants. En effet, pour chaque objet, du modèle réseau, hiérarchique, relationnel, ou objet, il est possible de construire l'arbre XML associé. Les nombreux standards associés à ce langage en font le candidat idéal comme modèle d'intégration dans une architecture d'accès à des données distribuées.

4. Architecture des schémas

Dans notre approche , le concept de vue est primordial. En effet, plusieurs utilisateurs du système, entre autre plusieurs applications utilisant le système, voudront voir la même donnée de manière différente. Dans notre architecture , nous adoptons l'approche *GAV,* car elle est plus simple à implémentée, et répond mieux aux exigences de notre système qui est de faire intégrer des GUI (*Graphique User Interface*) existantes.

Pour ce faire, on utilise un schéma virtuel intermédiaire , relationnel, qui servira pour la modélisation de la structure du document XML. En effet, les utilisateurs du système ne voient que la vue intégrée sur laquelle, ils envoient des requêtes SQL. Cependant, ces requêtes sont reformulées, afin d' être exécutées sur les sources de données relationnelles locales. Par conséquent, seulement les résultats de ces requêtes sont matérialisés. L'architecture que nous préconisons est décrite dans la figure 6.2.

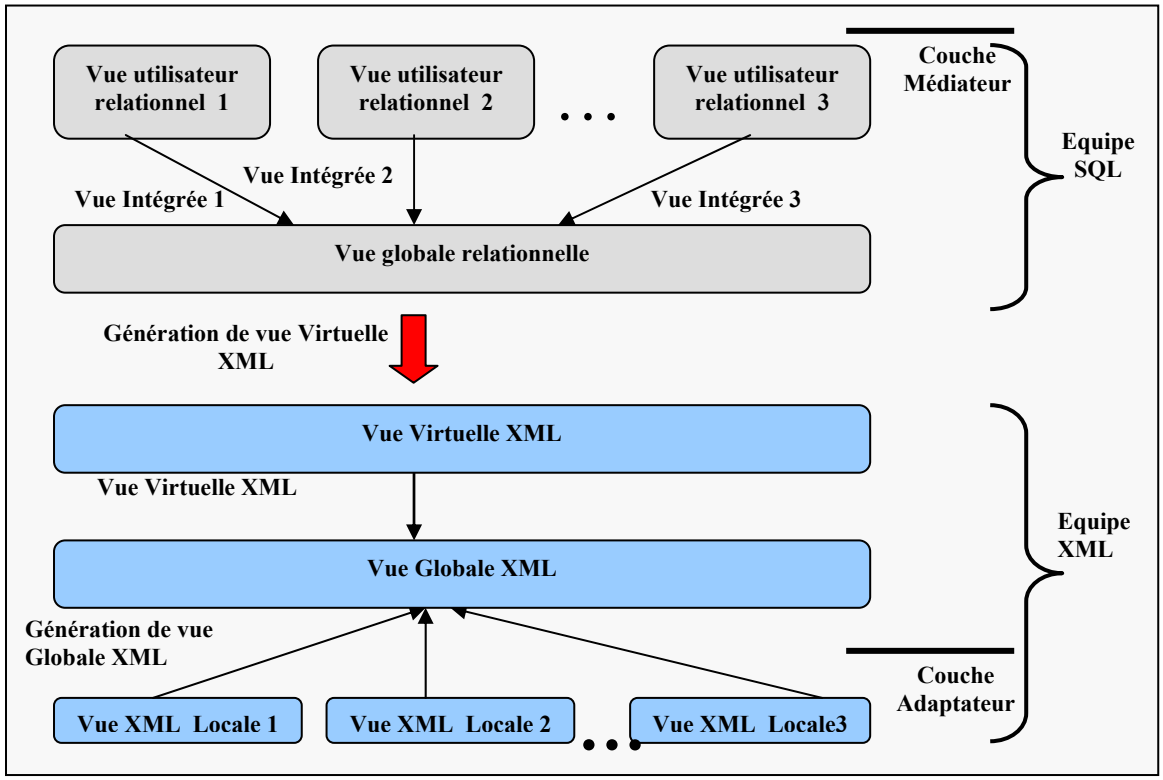

FIG. 6.2- Architecture des schémas d'une approche « tout-relationnel »

- *Vue XML locale (vue d'exportation) :* c'est une vue XML sur la source des données locale. Elle est exportée par l'adaptateur. Elle permet d'avoir un schéma XML des données existantes dans la source.

- *Vue XML globale :* cette vue est générée par le médiateur, elle représente l'union des différentes vues *XML locales*. Le médiateur garde une trace entre chaque vue XML locale et sa source d'origine.

- *Vue virtuelle XML (vue XML Utilisateur) :* cette vue est construite au dessus de *la vue globale XML.* Elle correspond aux besoins utilisateurs. Elle est générée automatiquement dans le médiateur à *partir de la vue globale relationnelle*. La vue virtuelle XML représente la vue relationnelle globale dans le format XML.

- *La vue globale relationnelle :* cette vue est conçue par l'équipe de développement SQL, elle représente le schéma global de la base de données fédérées. Elle est générique et contient toutes les informations sur les méta données du système. Elle permet de construire la vue XML Utilisateur.

- *Les vues utilisateurs relationnelles :* chacune de ces vues représente la vue utilisateur, c'est à dire , les schémas de données de chaque application qui participe à la fédération. elle facilite l'intégration de nouvelle application dans le système puisqu'il suffit d'intégrer la vue relationnelle sous jacente à l'application.

Quand une interface GUI utilisateur lance une requête *SQL* globale dans le système fédérés (figure 6.3). Cette requête est décrite sur *la vue relationnelle utilisateur,* elle est traduite en (1) , par le médiateur dans *la vue relationnelle globale.*

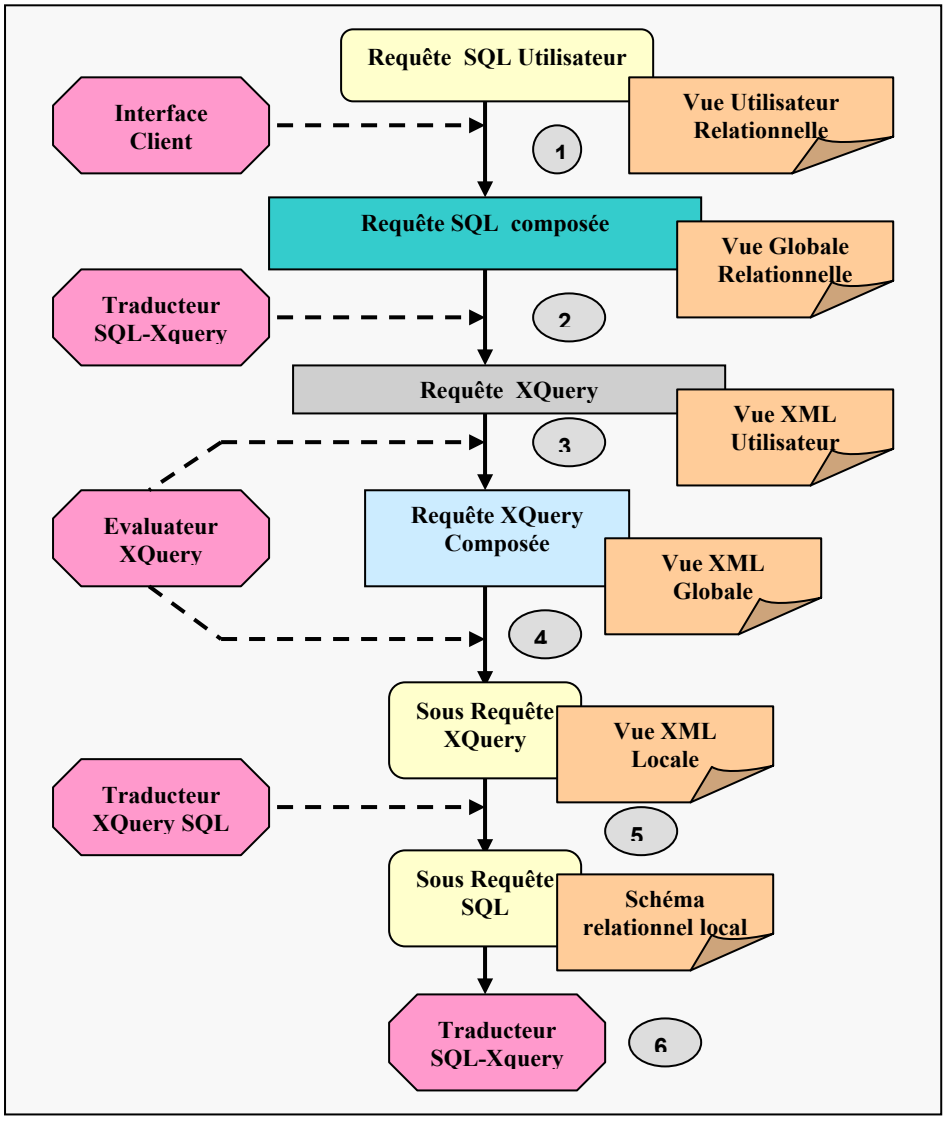

FIG. 6.3 - Phases d'évolution d'une requête dans une approche « Tout-Relationnel »

Puis la requête *SQL résultante* est traduite en (2) en une requête XQuery écrite dans *la vue XML utilisateur.* Des règles de traduction de vues sont ensuite appliquées en (3) pour que cette requête soit réécrite dans la *vue XML globale* ,qui est l'union des vues *XML locales.* On obtient une requête XQuery globale. Elle est décomposée en sous requêtes XQuery en (4). Chaque sous requête est envoyée ensuite à l'adaptateur de la source sous jacente. Elle sera traduite une dernière fois en (5) dans le modèle de la source puis envoyée au SGBD relationnel sous jacent (6).

5. L'exportation des données relationnelles en XML

Le premier composant dans notre architecture consiste dans l'adaptateur, il s'occupe de l'exportation des données relationnelles en XML. Cette fonction s'exécute en deux étapes (figure 6.4) :

- *Génération de schémas XML-schéma* : un schéma XML correspondant aux données relationnelles de la source qui est sous jacente à l'adaptateur est généré. Une table de *mapping* comportant les règles de traductions, est aussi élaborée. Lors de l'initialisation, *l'adaptateur* soumet au *médiateur* ces deux documents, générés par le *générateur de schéma.*

- *Composition de vues :* lors de l'évaluation d'une requête , l'adaptateur traduit la requête XQuery envoyée par le médiateur en requête SQL, la met à la disposition du moteur SGBD R sous jacent. Finalement, il retourne le résultat au médiateur. C'est le composant *traducteur de requêtes XQuery* qui exécute cette tache, il définit une vue d'exportation de manière déclarative, sur laquelle il compose les requêtes du médiateur, les traduisant ainsi en requêtes SQL ,qu'il envoie au moteur Relationnel. Le composant *Générateur de résultat*, quant à lui, permet de générer le document XML résultat à partir des tuples de données rendus par le SGBDR.

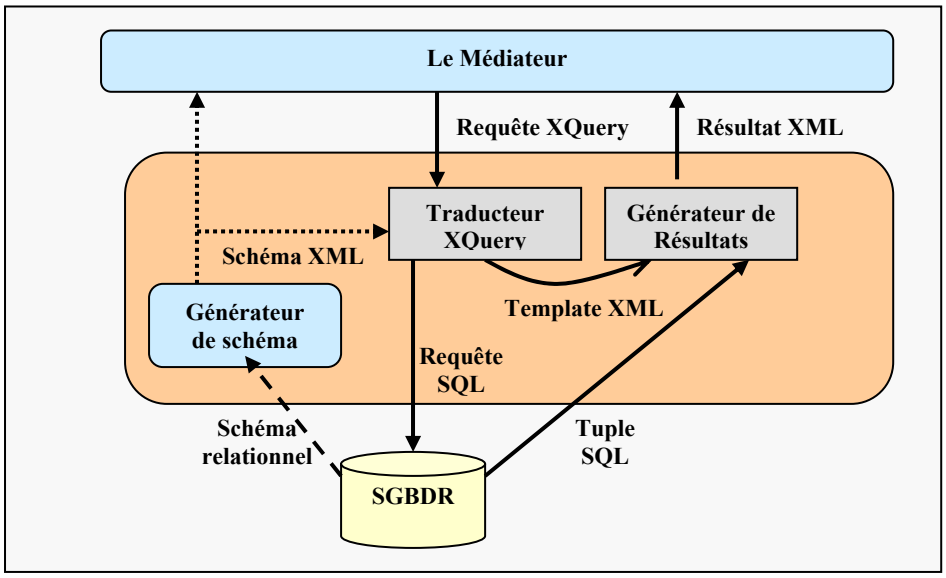

Fig. 6.4 – Architecture de l'adaptateur

5.1 Le générateur de schéma

L'adaptateur s'occupe, en premier lieu de l'exportation de la source de données qui lui est sous jacente grâce au composant *générateur de schéma*. Il exporte le schéma de la base de données relationnelle sous forme de schéma XML-Schéma. *Une table de mapping* est aussi générée, elle comporte les règles de mapping entre les deux schémas. Ce schéma une fois exportée , représente la *vue d'exportation XML* $\mathcal{V}$, ou *Locale***.**

Pour exporter des données relationnelles en XML, plusieurs algorithmes sophistiqués ont été développés ***[LEE 02a] [LEE 01] [LEE 02b] [MAN 00][POP 02]***. Dans notre travail, on

utilise un algorithme simple qui permet de répondre pleinement aux besoins du système. Il repose sur les deux heuristiques suivantes***[LEE 02a]*** :

1. On décide au départ de la relation R sur laquelle on va grouper les éléments ; elle sera traduite en un élément *Relation_R*. Les contraintes de clés étrangères sont supportées en incluant l'élément qui fait référence, dans la définition de type de l'élément référencé.

2. Convertir les relations de la base , en tenant compte de la contrainte précédente ,en un schéma XML-schéma ; relier les schémas issus de relations indépendantes dans un document unique qu'on appelle *collection.*

Pour illustrer les différentes techniques mises en œuvre , on prendra pour la suite du chapitre l'exemple de la source de données relationnelles *Etudiants* (figure 6.5):

Etudiant (*CodeEtudiant*, Nom, prénom)

Boursier (*CodeOeuvre* , CodeEtudiant, Montant)

Inscription(*CodeEtablissement*, CodeEtudiant, Filière, Moyenne, AnneePedag)

FIG. 6.5 – schéma relationnel de la source *Etudiants*

Ou La table *Etudiant* contient les informations sur les étudiants , *Boursier,* sur la bourse de chaque étudiant et *Inscription* sur son inscription. L'attribut *CodeEtudiant* dans les tables *Boursier* et *Inscription* est une clé étrangère, les clés primaires sont décrites en italique. La figure suivante (figure 6.6) illustre la forme qu'aura le document qui correspond aux tuples de la base de données *Etudiants* exportée selon l'approche citée si dessus .

```
<Relation_etudiant>
<Etudiant>
    <nom> … </nom>
    <Prénom> …      </Prénom>
    <Boursier> … </Boursier>
   <Inscription>…    </Inscription>
   <Inscription>…    </Inscription>
   <Inscription>…    </Inscription>
          …
</Etudiant>
……
</Relation  etudiant>
```

FIG. 6.6 – Exportation de la source *Etudiants* en XML

Les éléments *Inscription* et Boursier auront la forme suivante (figure. 6.7) :

```
<Inscription>
    <Etablissement>  …  </ Etablissement >
    <Moyenne d'accès>  …  </Moyenne d'accès >
     <Filière d'inscription>  …  </Filière d'inscription >
     <Annee>  …  </ Annee >
<Inscription>
```

```
<Boursier>
    <Montant Bourse> … </Montant Bourse>
    <CodeOeuvre> …</CodeOeuvre>
</Boursier>
```

FIG. 6.7 – Format des sous éléments *Inscription* et *Boursier*

Le schéma XML-schéma exporté est le suivant (figure 6.8): on définit, deux éléments *Etudiant* et *Inscription*, de type complexe, respectivement *TypeEtudiant* et *TypeInscription*. Chacun définit ses sous éléments, un sous élément pour chaque attribut de la relation.

```
‘ définition du type complexe TypeInscription
< xsd : ComplexType      name = ‘’TypeInscription’’  >
   <xsd : sequence >
         <xsd : element      name = “Etablissement”    type =”String” >
         <xsd : element      name = “Moyenne d’accès”    type =”Integer” >
         <xsd : element      name = “Filiere d’inscription”    type =”String” >
         <xsd : element      name = “AnneePedag”    type =”string” >
       </xsd : sequence >
</xsd : ComplexType>
 ‘ définition du type complexe TypeBoursier
< xsd : ComplexType      name = ‘’TypeBoursier’’  >
   <xsd : sequence >
         <xsd : element      name = “Montant Bourse”    type =”Integer” >
         <xsd : element      name = “Oeuvre”    type =”String” >
     </xsd : sequence >
</xsd : ComplexType>
‘ définition du type complexe TypeEtudiant
< xsd : ComplexType      name = ‘’TypeEtudiant’’  >
   <xsd : sequence >
         <xsd : element      name = “Nom”    type =”String” >
         <xsd : element      name = “Prenom”    type =”String” >
         <xsd : element      name = “Boursier”    type =”TypeBursier” >
         <xsd : element      name = “Inscription”    type =”TypeInscription” >
      </xsd : sequence >
</xsd : ComplexType>
‘ définition de  type relation_etudiant
<xsd : ComplexType  name = ‘’TypeRelation’’ >
    < xsd : element     name =”Etudiant     Type =”TypeEtudiant”>
</xsd : element>
‘Déclaration
<xsd : element name = ‘’Relation_Etudaint ‘’  tyoe =’’TypeRelation’’
```

FIG. 6.8 - le schéma XML-schéma utilisateur de la Base Etudiants

Dans le schéma relationnel de notre exemple, les deux relations *Boursier* et *Inscription* contiennent une référence (clé étrangère : *CodeEtudiant*) à relation *Etudiant*. Par conséquent dans le schéma XML correspondant, la relation E*tudiant* est convertie en un élément de type *TypeEtudiant.,* dans lequel on déclare les éléments *Boursier* et *Indcription* dont le type est respectivement *TypeBoursier* et *TypeInscription.*

La table de mapping permet de renommer les objets selon les besoins de l’intégration (de décider des données à exporter…etc). La table de mapping qui décrit le *mapping* précédent est illustrée dans la figure (figure 6.9). Les signes ← dans la table indiquent une réduction du membre gauche qui représente un des éléments du modèle relationnel vers le membre droit qui est un élément du modèle XML.

‘ réduction de la base en document ainsi que les tables
Etudiant ← collection(« Etudiant.XML »)
Boursier ← collection(« Etudiant.XML »)/Boursier
Inscription← collection(« Etudiant.XML »)/Inscription
‘ ajout d’un élément racine pour chaque relation
racine = relation_Etudiant
‘ nommage et définition des sous éléments de la racine
Nom ← Etudiants/Nom
Prénom ← Etudiant/Prénom
MontantBourse← Etudiant/Boursier/Montant Bourse
CodeOeuvre← Etudiant/Boursiet/Œuvre
CodeEtablissement ← Etudiant/Inscription/Etablissement
Moyenne ← Etudiant/Inscription/ Moyenne d’accèe
Filiere ← Etudiant/Inscription/ Filiere d’inscription
AnnePedag← Etudiant/Inscription/ Annee

FIG. 6.9– Table de mapping de la source *Etudiants*

Lors de son initialisation, l’adaptateur envoie le schéma XML-Schéma et la table de *mapping,* au médiateur pour les besoins d’intégration des données.

5.2 Le traducteur de requêtes XQuery

Le deuxième composant de l’adaptateur consiste dans *le traducteur de requêtes XQuery*. Sa fonction principale consiste dans la traduction de la requête XQuery envoyée par le médiateur, en une requête SQL, en utilisant la *Vue d’exportation XML* $\mathcal{V}$. générée dans l’étape précédente.

Pour ce faire, le traducteur utilise le mécanisme *de composition de vue****[ABI 99]***. En effet, la *Vue d’exportation XML* $\mathcal{V}$, n’est pas matérialisée, mais gardé en code source, sous forme de spécifications décrites de manière déclarative. Elle sert à la réécriture des requêtes dans le schéma local de la source. Les données effectives ne seront matérialisées que lors de l’exécution d’une requête.

Dans la littérature plusieurs langages de spécification de la vue d’exportation $\mathcal{V}$ existent (RXL de SkillRout ***[FRN 00]***, XQGM de Xperanto ***[SHA 01]*** ,UXquery ***[BRA 03b]***). Ils se basent essentiellement sur un formalisme qui combine les spécifications du langage XML utilisé et SQL dans deux blocs d’instructions. Dans notre cas, où le langage XML utilisé est XQuery, les deux blocs sont tel que :

- Un bloc SQL : contient les clauses FORM et WHERE qui permettent d’exporter le schéma des données relationnelles.
- Un bloc XQuery : contient une clause RETURN , qui permet de déclarer la syntaxe XQuery du document XML résultat souhaité.

La vue $\mathcal{V}$, vue dans l’exemple précédent, est déclarée comme suit :

```
V = FROM  Etudiant   $Etud
     RETURN
      <Relation_Etudiant>
        <Etudiant>
       <nom>  $ Etud. nom </nom>
       <Prénom> $ Etud.Prénom   </Prénom>
       (FROM Boursier  $Bour
       WHERE $Bour.CodeEtudiant = $Etud.CodeEtudiant
        RETURN
     <Boursier>
        <Montant Bourse> $Bour.MontantBourse</Montant Bourse>
        <Œuvre >$Bour.MontantBourse </Œuvre >
     </Boursier>
     (FROM  Inscription  $Inscri
     WHERE   $Etud.CodeEtudiant =  $Inscri.CodeEtudiant
    RETURN
     <Inscription>
     <Moyenne d'accès>  $Inscri.Moyenne</Moyenne d'accès>
      <Filière d'inscription> $Inscri.Filiere</Filière d'inscription >
      <Annee> $Inscri.AnneePedag</Annee>
     < Etablissement> $ Iscrit.CodeEtablissement </Etablissement>
          </Inscription>
         )
        </Etudiant>
   </Relation_Etudiant>
```

La traduction d'une requêtes XQuery vers SQL s'effectue à travers les étapes suivantes :

- ***Etape 1 :*** Quand l'adaptateur reçoit une requête XQuery, *le traducteur de requêtes* la compose avec $\mathcal{V}$, et génère une nouvelle requête $\mathcal{Req}$ composée de deux blocs d'instructions *SQL* et *XQuery.*

- ***Etape 2 :*** Le *traducteur* décompose la requête $\mathcal{Req}$ en deux blocs :
  - Un bloc d'extraction de données : une ou plusieurs requêtes SQL, qui sont dirigées vers le moteur relationnel de la source de données.
  - Un bloc XML pour la construction du résultat XML ; qui est dirigé vers le composant *générateur XML.*

- ***Etape 3 :*** après exécution de la requête SQL, le moteur relationnel remet au *générateur XML* les tuples résultant, celui ci les traduit en documents XML selon la structure XML fournie.

Pour illustrer ce fonctionnement, supposons , que le médiateur envoie la requête suivante à l'adaptateur :

« Req : Obtenir les noms des établissements dans lesquels les étudiants ont une bourse supérieure à 200 DA »

Cette requête est écrite en XQuery :

```
For $Var  in  V
Where $var /Montant Bourse > 200
Return <Nom Etablissement> $var/Etablissement<Nom Etablissement>
```

Alimenté avec cette requête , le traducteur va la *réécrire.* Cela veut dire la composer avec $\mathcal{V}$ et elle deviendra $\mathcal{Req}$:

```
For $Var in (FROM  Etudiant   $Etud
     RETURN
      <Relation_Etudiant>
        <Etudiant>
       <nom>  $ Etud. nom </nom>
       <Prénom> $ Etud.Prénom   </Prénom>
       (FROM Boursier  $Bour
       WHERE $Bour.CodeEtudiant = $Etud.CodeEtudiant
        RETURN
     <Boursier>
        <Montant Bourse> $Bour.MontantBourse</Montant Bourse>
        <Œuvre >$Bour.MontantBourse </Œuvre >
     </Boursier>
     (FROM  Inscription  $Inscri
     WHERE   $Etud.CodeEtudiant =  $Inscri.CodeEtudiant
    RETURN
     <Inscription>
     <Moyenne d'accès>  $Inscri.Moyenne</Moyenne d'accès>
      <Filière d'inscription> $Inscri.Filiere</Filière d'inscription >
      <Annee> $Inscri.AnneePedag</Annee>
     < Etablissement> $ Iscrit.CodeEtablissement </Etablissement>
          </Inscription>
         )
        </Etudiant>
   </Relation_Etudiant>
)
WHERE $var/Montant Bourse >200
RETURN <Nom Etablissement> $var/Etablissement<Nom Etablissement>
```

Dans la requête $\mathcal{Req}$, on procède à des simplifications et à des manipulations algébriques, parmi lesquelles, on cite :

- Correspondance des variables : dans la requête XQuery , les variables *$Var/Montant bourse* et *$Var/Etablissement* représentent respectivement le contenu des éléments <montant bourse> et <Etablissement> , elles sont donc remplacées par *$Bour.MontantBourse* et *$Etud.CodeEtablissement.*

- Le résultat final des deux requêtes composées est l'intersection de leurs différentes clauses RETURN .

Ainsi après simplification de la requête précédente, on obtient la requête suivante :

```
FROM  Etudiant   $Etud , Boursier  $Bour, Inscription $Inscri
WHERE $Bour.CodeEtudiant = $Etud.CodeEtudiant AND
          $Bour.montantBourse >200  AND
         $Etud.CodeEtudiant =  $Inscri.CodeEtudiant
RETURN
 <Nom Etablissement> $Inscri.CodeEtablissement <Nom Etablissement>
```

Cette requête peut être évaluée par un moteur relationnel à l'exception de la partie RETURN qui sera envoyée au *générateur de résultat.*

5.3 Le Générateur de résultats XML

Ce composant exporte les tuples relationnels qu'il reçoit en entrée, en document XML. A Chaque tuple, il lui applique une *balise* ,selon *la template* XML décrite dans la clause RETURN de la requête. Il faut que la traduction de la requête et la génération de résultats soient effectués en une seule passe.

6. La médiation entre sources hétérogènes

Le médiateur est un élément clé dans notre architecture, grâce à lui, l'utilisateur peut continuer à avoir l'illusion du « tout-Relationnel ». Ce composant (figure 6.10) est construit autour d'une architecture de schémas capable de traiter des requêtes SQL utilisateur et de renvoyer des tuples relationnels, en utilisant une infrastructure basée sur le modèle XML et le langage XQuery.

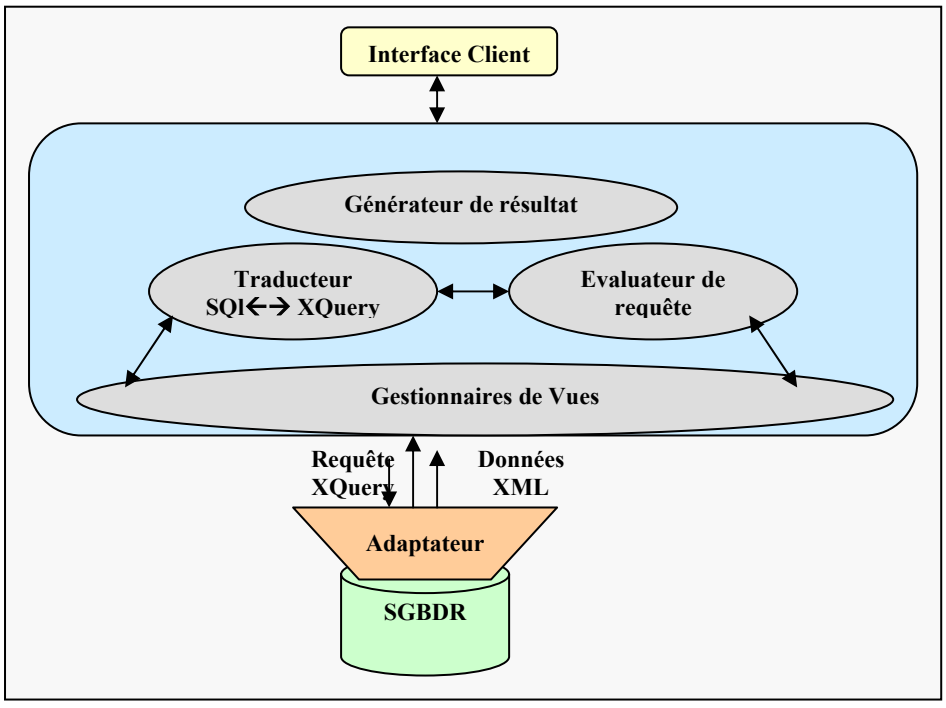

FIG.6.10 – Architecture du médiateur

6.1 Le gestionnaire des vues

Le but de notre système est de permettre l'exploitation de plusieurs sources de données indépendantes comme s'il s'agissait d'une source unique, avec un schéma global unique. La requête utilisateur est formulée en terme du schéma global ; pour exécuter une requête , le système la traduit en sous requêtes exprimées sur les schémas locaux, envoie les sous requêtes aux sources locales, obtient les résultats, et les combine dans le résultat fournit à l'utilisateur.

Le gestionnaire de vue permet d'implémenter ce modèle de vues, qui est capable de prendre en charge les spécificités de notre système, à savoir, la traduction SQL-> XQuery ainsi que l'architecture qui lui est associée. Cette architecture des schémas permet de présenter à l'utilisateur une vue uniforme et centralisée des données distribuées. Dans ce qui suit , nous décrirons les principales techniques mises en œuvre :

6.1.1 Fonctionnement

Le gestionnaire de vue est un composant complexe qui permet de mettre en œuvre une infrastructure de définition de vues semblable à celle utilisée dans l'adaptateur. Il effectue entre autre trois fonctions majeures :

- Il récolte les différentes schémas- XML exportés par les adaptateurs , et construit la *vue globale XML.*
- Il reçoit en entrée *la vue globale relationnelle*, qui sert de schéma d'exécution au requêtes SQL utilisateurs. Et a partir de laquelle, il construit *la vue XML Utilisateur* au dessus de *la vue XML Globale.*
- Il alimente, finalement, *l'évaluateur de requêtes* et le *traducteur SQL XQuery* avec les différents schémas de données nécessaires à leur exécution.
- Composer la requête utilisateur SQL sur la vue relationnelle globale.

6.1.2 Génération de la vue globale XML

La première étape dans la médiation, consiste dans la récolte d'information à partir des adaptateurs. En effet, lors de l'initialisation, chaque adaptateur envoie les informations de *méta données (schéma exporté)* de la source qui lui est associée. Le médiateur (*gestionnaire de vues*) fusionne ensuite cette description de méta données sous forme de *vue XML globale,* en gardant trace des caractéristiques de chaque source (adaptateur associé, nom de la collection).

On reprend la base *Etudiants*. Supposons que cette base fait partie d'un système fédéré comportant deux autres bases relationnelles, la base *Bac* décrivant tous les bacheliers et située sur le *site 1 ;* et la base *Résidence* située sur le site *Site 2.* de la même manière que pour la base *Etudiant,* les adaptateurs sous jacents aux autres bases vont exporter les schémas XML-schéma. Respectivement : *schema-etudiant* , *schema-bac* et *schema-residence.* Les trois adaptateurs sont décrits dans la figure (6.11) .

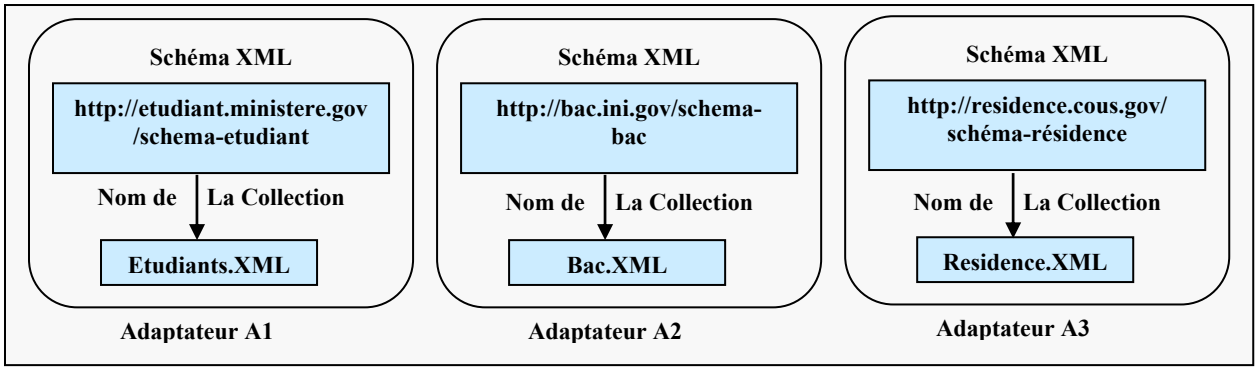

FIG. 6.11 – description de schémas au niveau adaptateur

Au niveau du *gestionnaire de vues* (figure 6.12) , *la vue globael XML* est définie , c'est une vue virtuelle, stockée au niveau du *cache du gestionnaire de vues*. Elle correspond à l'union des schémas de *A1, A2, A3.* Des informations supplémentaires sur l'appartenance (adaptateur) de chaque schéma sont gardées dans le *gestionnaire de vues.*

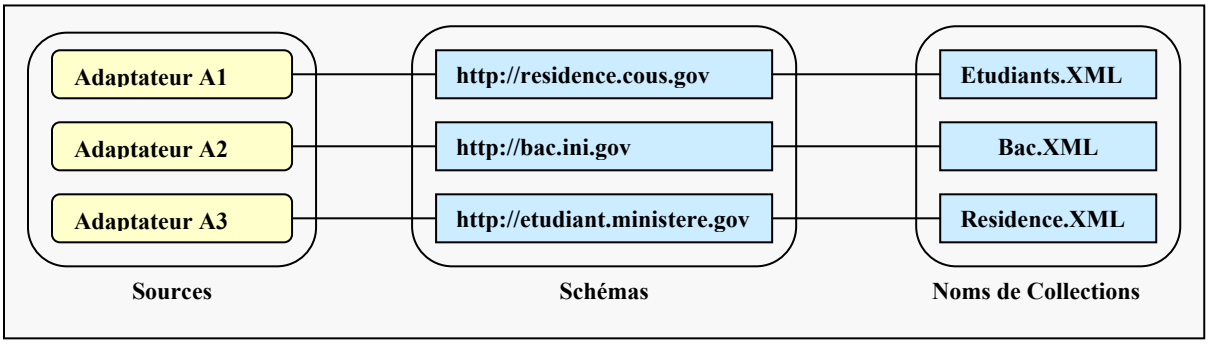

FIG. 6.12 – description des schémas au niveau Médiateur

On définit en pseudo code , la fonction *Import* qui permet d'importer les schémas.

```
Vue_Globale_XML = (

Vue_Globale_XML / Etudiant : = Import (schéma-Etudiant)
Vue_Globale_XML / Etudiant : = Import (schéma-Résidence)
Vue_Globale_XML / Etudiant : = Import (schéma-Bac)
)
```

On peut manipuler les schémas de la vue globale en utilisant les fonctions *GetEtudiant*, *GetRésidence* et *GetBac* .

```
GetEtudiant()  { return  Vue_Globale_XML / Etudiant  }
GetRésidence()  { return  Vue_Globale_XML / Résidence  }
GetBac()  { return  Vue_Globale_XML / Bac  }
```

Pour résoudre le problème de redondance de vocabulaires lors de la définition de la *vue XML globale,* de utilise un espace de nom différent pour chaque schéma :

```
....
Xmlns : xse ='' http://residence.cous.gov/Etudiant-schema ''
Xmlns : xsb = '' http://bac.ini.gov/ bac-schema''
Xmlns : xsr ='' http://etudiant.ministere.gov/schéma-residence ''
...
```

6.1.3 Génération de la vue XML Utilisateur

L'idée de base dans la construction de la vue *XML Utilisateur* ,est d'être dirigée par *la vue relationnelle globale ;* de sorte que les éléments décrits par la *vue XML Utilisateur* auront *la forme* décrite dans le *schéma XML* correspondant de *la vue globale relationnelle.* Ceci implique , que les équipes de développement SQL se concentrerons sur la conception du *schéma global* du système fédéré, ce schéma global sera utilisé par le médiateur comme *template* pour la génération de la *vue XML utilisateur.*

Donc, la deuxième étape dans le développement du médiateur consiste, dans la définition de la *vue relationnelle globale.* Cette vue est complètement virtuelle, elle sert à *structurer* la vue *XML Utilisateur* ; Elle correspond au schéma global et centralisé des données du système fédéré. Lors de la configuration du médiateur, on lui soumet *la vue globale relationnelle*. Celui ci génère le *schéma XML-schéma* correspondant suivant la technique décrite dans la section (exportation de données relationnelles en XML). Appelons ce schéma *Schéma-Intermédiaire.*

A partir de ces données , le *gestionnaire de vues* définit la *vue XML Utilisateur* de manière déclarative , elle sera construite lors de l'exécution d'une requête de manière dynamique en utilisant le principe de composition de vues***[BRA 03a] [BRA 03b] [GRU 04]***. *La vue XML Utilisateur*, qu'on appellera *V_User* est définie par l'algorithme suivant :

```
V_User =  Générer_ VueXML_User   ( Vue XML Globale, Schéma-Intermédiaire)
    {
    for $var in Vue XML Globale
    return
    Schéma-Intermédiaire o ( <$var>   ....      </$var>)
    }
```

La fonction décrite construit une *vue XML utilisateur V_User,* dont les éléments auront la forme décrite par la syntaxe du schéma *Schéma-Intermédiaire* , qui est le schéma XML de la vue relationnelle globale. Implémenter cette fonction revient à résoudre le problème de correspondance entre les entités des deux différentes vues.

Illustrons ce mécanisme par l'exemple précédent , la figure 6.13. présente l'ensemble des schémas relationnels des trois sources citées précédemment :

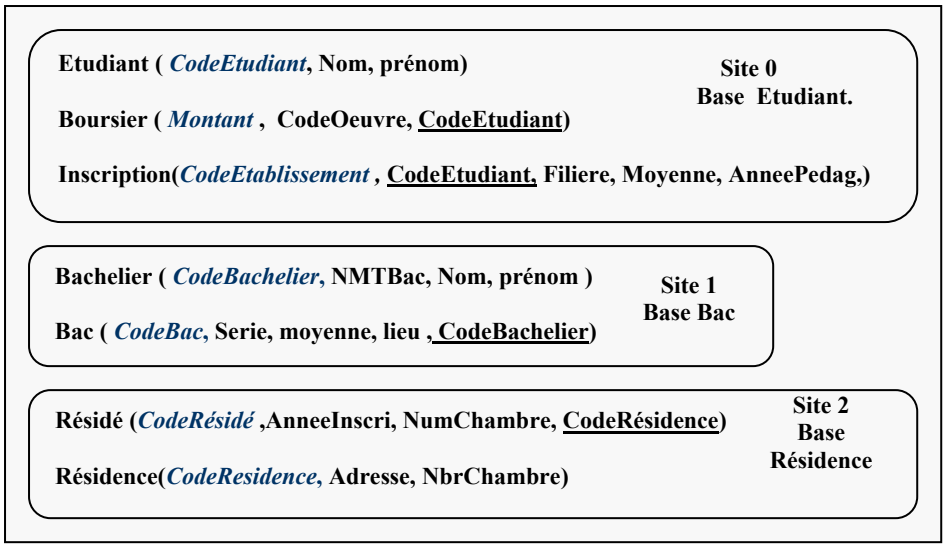

FIG. 6.13 – les schémas relationnels des sources Etudiants, Bac et Résidence

Leurs schémas XML correspondants sont décrits en *(*figure 6.14. a, b, c).

```
‘ définition du type complexe TypeInscription
< xsd : ComplexType      name = ‘’TypeInscription’’  >
   <xsd : sequence >
         <xsd : element      name = “Etablissement”    type =”String” >
         <xsd : element      name = “Moyenne d’accès”    type =”Integer” >
         <xsd : element      name = “Filiere d’inscription”    type =”String” >
         <xsd : element      name = “AnneePedag”    type =”string” >
       </xsd : sequence >
</xsd : ComplexType>
‘ définition du type complexe TypeBoursier
< xsd : ComplexType      name = ‘’TypeBoursier’’  >
   <xsd : sequence >
         <xsd : element      name = “Montant Bourse”    type =”Integer” >
         <xsd : element      name = “Oeuvre”    type =”String” >
     </xsd : sequence >
</xsd : ComplexType>
‘ définition du type complexe TypeEtudiant
< xsd : ComplexType      name = ‘’TypeEtudiant’’  >
   <xsd : sequence >
         <xsd : element      name = “Nom”    type =”String” >
         <xsd : element      name = “Prenom”    type =”String” >
         <xsd : element      name = “Boursier”    type =”TypeBursier” >
         <xsd : element      name = “Inscription”    type =”TypeInscription” >
       </xsd : sequence >
</xsd : ComplexType>
‘ définition de  type relation_etudiant
<xsd : ComplexType  name = ‘’TypeRelation’’ >
     < xsd : element     name =”Etudiant     Type =”TypeEtudiant”>
</xsd : element>
‘Déclaration
<xsd : element name = ‘’Relation_Etudaint ‘’  tyoe =’’TypeRelation’’
```

FIG.6.14 (a) - le schéma Schéma_Etudiant

```
‘ définition du type complexe TypeBac
< xsd : ComplexType      name = ‘’TypeBac’’   >
   <xsd : sequence >
         <xsd : element      name = ‘’Serie”    type =”String” >
         <xsd : element      name = “Moyenne”    type =”Integr” >
         <xsd : element      name = “Lieu”    type =”String” >
       </xsd : sequence >
</xsd : ComplexType>
‘ définition du type complexe TypeBachelier
< xsd : ComplexType      name = ‘’TypeBachelier’’   >
   <xsd : sequence >
         <xsd : element      name = “NmtBac”    type =”Integer” >
         <xsd : element      name = “nom”    type =”string” >
         <xsd : element      name = “Prénom”    type =”string” >
         <xsd : element      name = “Bac”    type =”TypeBachelier” >
       </xsd : sequence >
</xsd : ComplexType>
‘ définition de  type relation_bac
<xsd : ComplexType  name = ‘’TypeRelation’’ >
     < xsd : element     name =”bachelier’’    Type =”TypeBachelier”>
</xsd : element>
‘Déclaration
<xsd : element name = ‘’Relation_bachelier ‘’  tyoe =’’TypeRelation’’
```

FIG.6.14 (b) - le schéma Schéma_Bac

```
 ‘ définition du type complexe TypeRésidé
< xsd : ComplexType      name = ‘’TypeRésidé’’   >
   <xsd : sequence >
         <xsd : element      name = “CodeRésidé”  type =”String” >
         <xsd : element      name = “NumChambre”    type =”String” >
         <xsd : element      name = “anneeInscri”    type =”Integer” >
       </xsd : sequence >
</xsd : ComplexType>
‘ définition du type complexe TypeRésidence
< xsd : ComplexType      name = ‘’TypeRésidence’’   >
   <xsd : sequence >
         <xsd : element      name = “CodeRésidence”    type =”Integer” >
         <xsd : element      name = “Adresse”   type =”string” >
         <xsd : element      name = “NbeChambre”    type =”string” >
         <xsd : element      name = “Résidé”    type =”ITypeRésidé” >
       </xsd : sequence >
</xsd : ComplexType>
‘ définition de  type relation_bac
<xsd : ComplexType  name = ‘’TypeRelation’’ >
     < xsd : element     name =”boursier’’    Type =”TypeBoursier”>
</xsd : element>
‘Déclaration
<xsd : element name = ‘’Relation_Boursier ‘’  tyoe =’’TypeRelation’’ >
```

FIG.6.14 (c) - le schéma Schéma_Résidence

D’autre par, le gestionnaire de vues est muni de *la vue globale relationnelle* telle qu’elle était conçue par les équipes SQL (figure 6.15) :

Etudiant (*CodeEtudiant*, Nom, prénom)

Boursier (*CodeOeuvre* , CodeEtudiant, Montant)

Inscription(*CodeEtablissement*, CodeEtudiant, Filière, Moyenne, AnneePedag)

Bac (*CodeBac*, NMTBac,Serie,moyenne, lieu , CodeEtudiant)

Résidence(*CodeRésidé* , CodeResidence , NumChambre, AnneeInscri, CodeEtudiant)

FIG. 6.15 – La vue relationnelle Globale

Les clés primaires sont en italiques, alors que les clés étrangères sont soulignées. Le *gestionnaire de vues* génère le schéma *XML-schéma intermédiaire* correspondant (figure 6.16).

```
‘ définition du type complexe TypeRésidence
< xsd : ComplexType      name = ‘’TypeRésidence’’ >
   <xsd : sequence >
         <xsd : element      name = “CodeRésidence”    type =”Integer” >
         <xsd : element      name = “NumChambre”    type =”string” >
         <xsd : element      name = “anneeInscri”    type =”Integer” >
       </xsd : sequence >
</xsd : ComplexType>
‘ définition du type complexe TypeBac
< xsd : ComplexType      name = ‘’TypeBac’’   >
   <xsd : sequence >
         <xsd : element      name = “NmtBac”    type =”Integer” >
         <xsd : element      name = ‘’Serie”    type =”String” >
         <xsd : element      name = “Moyenne”    type =”Integr” >
         <xsd : element      name = “Lieu”    type =”String” >
       </xsd : sequence >
</xsd : ComplexType>
‘ définition du type complexe TypeInscription
< xsd : ComplexType      name = ‘’TypeInscription’’   >
   <xsd : sequence >
         <xsd : element      name = “CodeEtablissement”    type =”String” >
         <xsd : element      name = “Moyenne”    type =”Integer” >
         <xsd : element      name = “Filiere”    type =”String” >
         <xsd : element      name = “AnneePedag”    type =”string” >
       </xsd : sequence >
</xsd : ComplexType>
‘ définition du type complexe TypeEtudiant
< xsd : ComplexType      name = ‘’TypeEtudiant’’   >
   <xsd : sequence >
         <xsd : element      name = “Nom”    type =”String” >
         <xsd : element      name = “Prenom”    type =”String” >
         <xsd : element      name = “MontantBourse”    type =”Integer” >
         <xsd : element      name = “CodeOeuvre”    type =”String” >
        < xsd : element     name =”Résidence”      Type =”TypeRésidence”>
         < xsd : element     name =”Bac”      Type =”TypeBac”>
         <xsd : element      name = “Inscription”    type =”TypeInscription” >
</xsd : sequence >
</xsd : ComplexType>
‘ déclaration de  la collection
    …
```

Fig. 6.16 - le schéma XML-schéma intermédiaire.

Les donnée auront donc la forme suivantes (figure 6.17)

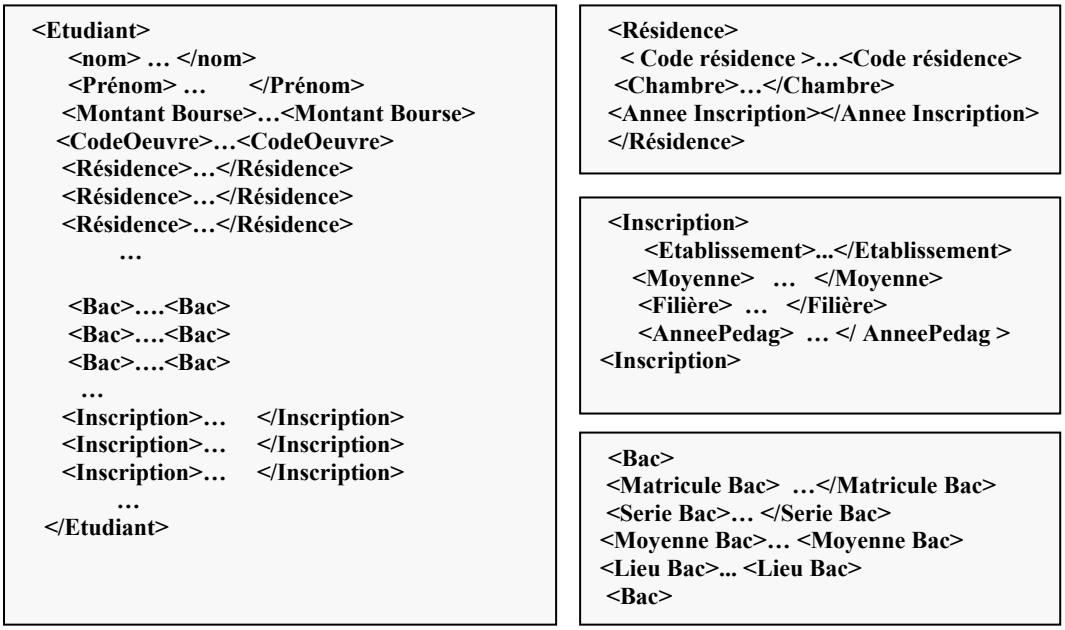

FIG. 6.17 – Forme des données modélisées par *le schéma intermédiaire*

Ce schéma servira de *template* pour la construction de la *vue XML utilisateur*. La construction de La *vue V_User* sera décrite en pseudo code comme suit :

```
V_User= ( For  $Etud   in  vue_globale_XML .GetEtudiant()
        Return
        <Relation_Etudiant>
         <Etudiant>
         <nom>  $ Etud/ nom </nom>
         <Prénom> $ Etud/Prénom   </Prénom>
         <Montant Bourse> $Etud/MontantBourse</Montant Bourse>
         <CodeOeuvre> $ Etud/CodeOeuvre </CodeOeuvre>
         ( For     $Resi  in vue_globale_XML .GetRésidence()
         WHERE   $Etud/CodeEtudiant =  $ Resi/Résidé/CodeRésidé
          RETURN
         <Résidence>
            <Code Résdeince> $Resi / CodeRésidence  </Code Résdeince>
            <Chambre >  $Resi / NumChambre  </Chambre >
            <Annee Inscription>$Resi / AnneeInscri </Annee Inscription>
          </Résidence>        )
          ( For $bac Vue_Globale_XML . GetBac()
          Where $bac / codebac = $Resi / CodeRésidé
          RETURN
          <Bac>
              <Matricule Bac>  $bac /NMT   </Matricule Bac>
              <Serie Bac>$bac /bac/seriebac </Serie Bac>
              <Moyenne Bac>$bac /bac/Moyenne <Moyenne Bac>
```

```
            <Lieu Bac>$bac /bac/Lieu <Lieu Bac>
           </Bac>          )
          (  For $Inscri  in Vue_Globale_XML . GetEtudiant()/Inscription
          WHERE   $Etud.CodeEtudiant =  $Inscri.CodeEtudiant
          RETURN
           <Inscription>
              <Etablissement> $Inscri.CodeEtablissement </Etablissement>
              <Moyenne accès >  $Inscri.Moyenne</Moyenne accès >
             <Filière> $Inscri.Filiere</Filière>
          <AnneePedag> $Inscri.AnneePedag</AnneePedag>
           </Inscription>
)
</Etudiant>
</Relation  Etudiant>
```

6.2 Le traducteur SQL → XQuery

Muni du *schéma-intermédiaire* correspondant à la vue globale relationnelle, ce composant traduit les requêtes SQL en requêtes XQuery. (figure 6.18).

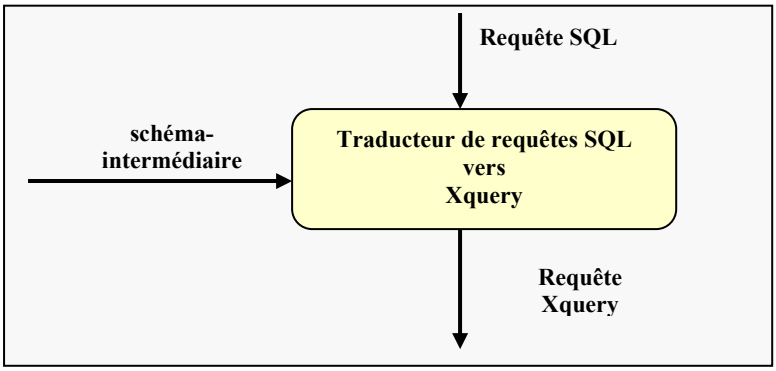

FIG. 6.18 – Traducteur de requête SQL e n XQuery

Le traducteur de requêtes SQL vers XQuery est un programme qui traduit la requête écrite dans le langage *source* , dans notre cas SQL , en une requête équivalente écrite dans le langage cible, à savoir *XQuery*.

SQL est le langage standard des bases de données relationnelles. Dans la plupart des cas, les requêtes SQL peuvent être reformulées dans la syntaxe XQuery de manière directe en faisant correspondre (*mapping)* les blocs de requêtes SQL en des expressions FLWR. ***[XQ 03]***

Les phases d'implémentation du traducteur s'inspire de la construction d'un compilateur , plus précisément, *de la traduction dirigée par la syntaxe.* Le processus de traduction décrit dans notre travail consiste dans les phases logiques suivantes ***[SIL 90]*** (figure 6.19):

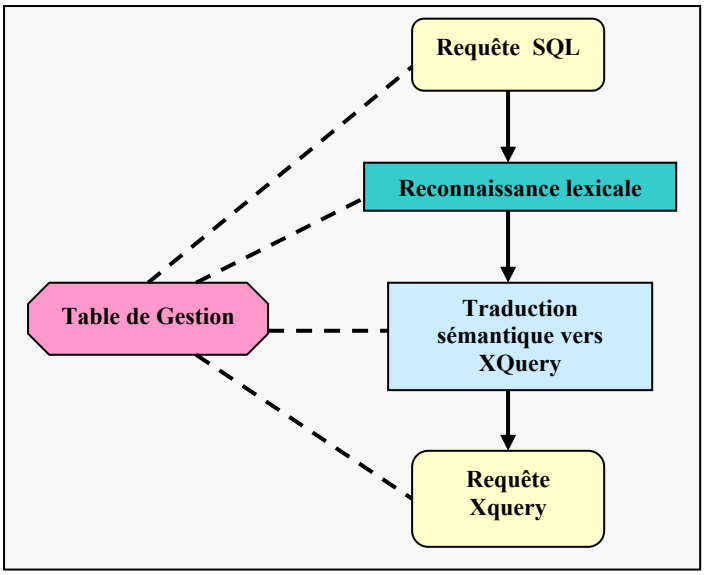

FIG. 6.19 – Structure Logique du traducteur

1) La Reconnaissance des entités lexicales

Durant cette phase, on constitue les mots de la requête SQL à partir de la chaîne de caractère de la requête. Selon la version de SQL qu'on veut utiliser ,on classifie les *mots* comme suit :

- Les identificateurs : représentent les mots formés par l'utilisateur pour définir les *noms de tables,* participant à la requêtes , les *nom des variables.*
- Les mots réservés du langage : les différentes clauses : SELECT, WHERE, FROM, GROUP BY, INSERT,…etc.
- Les constantes.
- Les opérateurs et les séparateurs : +, -,…et « ' ' » , etc.

Une *table de gestion* est générée à l'issue de cette étape, elle contient tous les identificateurs et les constantes de la requête.

2) Traduction dirigée par la syntaxe

De manière parallèle avec l'analyse syntaxique, *le traducteur de requête* procède à la traduction effective des requêtes. Pour ce faire il utilise deux techniques suivantes :

2.a) La résolution des chemins

Vue la structure arborescente des documents XML, notre traducteur est doté d'une fonctionnalité supplémentaire : la résolution de chemin des variables. On définit la fonction *résolution_chemin()* qui à partir de la *table de mapping* de l'exportation retourne le chemin d'un élément relationnel donné :

Fonction Résolution_Chemin (Table_mapping, Element_relationnel)
(
a partir de table_mapping retourner l'expression de chemin de l'élément Element_relationnel.
)

Pour illustrer cette fonctionnalité, on suppose que les relations sont exportées de manière plate vers un document XML. Considérons une partie du schéma relationnel , vu précédemment (la figure 6.20)

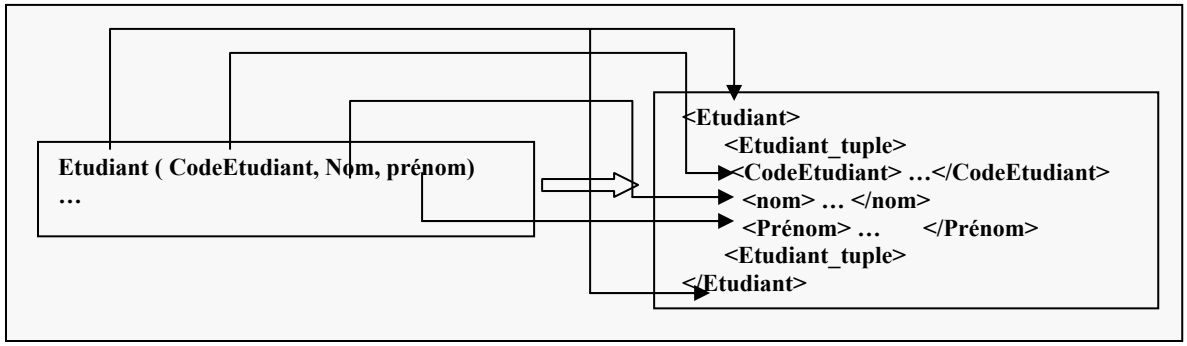

FIG. 6.20 – Correspondance entre schéma relationnel et XML

La relation *Etudiant* est représentée en XML par le schéma XML-schéma *Etudiant-schéma.* Dont la table de mapping correspondante *map_table* est décrite comme suit :

' réduction de la base en document ainsi que les tables
Etudiant ← collection(« Etudiant.XML »)
' Nommage et définition des sous éléments de la racine
CodeEtudiant ← Etudiant_tuple/CodeEtudiant
Nom ← Etudiant_tuple//Nom
Prénom ← Etudiant_tuple /Prénom

Considérons la requête suivante : « trouver les codes des étudiants dont le nom est Nessrine. » la figure 6.21. présente les versions SQL et XQuery de cette requête.

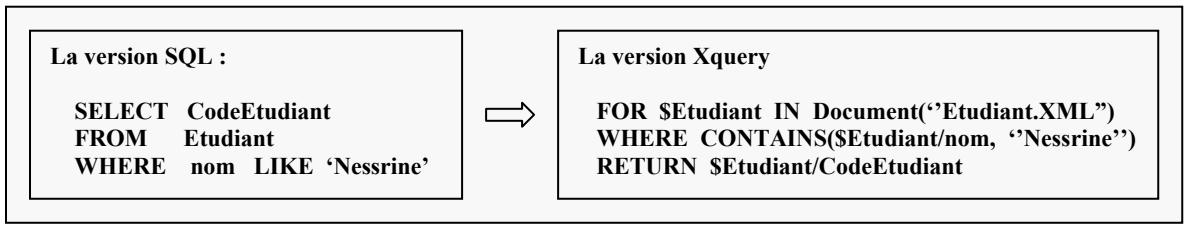

FIG. 6.21 - Traduction SQL vers XQuery

Dans l'exemple cité ci dessus, les éléments *CodeEtudiant* et *nom* qui correspondent aux attributs *CodeEtudiant* et *nom* de la relation R dans la requête SQL, peuvent se trouver à une profondeur quelconque du nœud racine *Etudiant* dans le schéma XML. On utilise alors la fonction *Résolution_Chemin* tel que :

Résolution_Chemin (map_table, ''CodeEtudiant '') = ***Etudiant_tuple/CodeEtudiant.***
Résolution_Chemin (map_table, ''nomt '') = ***Etudiant_tuple//nomt.***

2.b) Le Schéma de traduction dirigée par la syntaxe

Un schéma de traduction dirigé par la syntaxe est un formalisme qui comporte la grammaire du langage SQL, ainsi que des actions sémantiques. Les actions sémantiques sont des traductions attachées aux règles de la grammaire SQL. Pour illustrer la technique utilisée, nous nous appuyons sur un sous ensemble de la grammaire de SQL2 ***[SQL2]*** (figure 6.22).

```
1.  <query specification> ::= SELECT <column name> <table expression>
2.  <table expression>::= <from clause> <where clause>
3.  <from clause> ::= FROM <table name>
4.  <where clause> ::= WHERE <search condition>
5.  <column name> :: = <Identifier>
6.  <table name> ::= <identifier>
7.  <search condition> ::=<Column name> LIKE '<identifier>'
8.  <Identifier> := <Letter><letter>*
9.  <letter> := a | b | c | d | e | f | g | h | i | j | k | l | m | n | o | p | q | r | s | t | u | v | w | x | y | z
```

FIG. 6.22 - Grammaire d'un sous ensemble du langage SQL

Cette grammaire permet d'exprimer des requêtes SQL de la forme :

SELECT *nomColonne*
FROM *nomTable*
WHERE *NomColonne0 LIKE 'Chaîne de caractère'*

Pour construire le schéma de traduction de la grammaire citée ci dessus, on définit une action sémantique qui associe une traduction à chaque entité du langage SQL. Pour une entité donnée *Entité,* une traduction est définie par *Entité.tradu.* ainsi le schéma de traduction associé à la grammaire (figure 6.22) est illustré a la figure (6.23).

```
1.  <query specification> ::= SELECT <column name> <table expression>
    query_spec.tradu = return column_name.tradu table_expression.trad
2.  <table expression>::= <from clause> <where clause>
    teblae_xpression.tradu = from_clause.tradu where_clause.tradu
3.  <from clause> ::= FROM <table name>
    from_clause.tradu = For $var In table_name.tradu
4.  <where clause> ::= WHERE <search condition>
    where_clause.tradu = where search_condition.tradu
5.  <column name> :: = <Identifier>
    colmun_name.tradu = résolution_chemin ($var,identifier.tradu)
6.  <table name> ::= <identifier>
    table_name.tradu = résolution_chemin(document(),identifier.tradu)
7.  <search condition> ::=<Column name> LIKE '<identifier>'
    search_condition.tradu = CONTAINS ( column_name.tradu, 'identifier.tradu')
8.  <Identifier> := <Letter><letter>*
9.  <letter> := a | b | c | d | e | f | g | h | i | j | k | l | m | n | o
  | p | q | r | s | t | u | v | w | x | y | z
```

FIG. 6.23 - Traduction dirigée par la syntaxe

Ainsi, à la ligne 1 de la figure 6.23. , qui décrit la syntaxe d'une requête de Sélection , on lui associe la traduction équivalente dans le langage XQuery *;* ainsi SELECT est traduite en RETURN. Dans le schéma de traduction , on remarque la fonction *Résolution_schéma* décrite précédemment qui retourne le chemin exacte de l'élément.

l'implémentation du traducteur nécessite le développement d'un analyseur syntaxique. A cet analyseur, on incorpore les actions sémantiques selon le schéma de traduction. Les requêtes XQuery fournies auront la forme (Return ,Where, For), doivent être réorganisées sous la forme (For, Where, Return).

6.3 L'évaluateur des requêtes

Comme notre architecture repose entièrement sur les SGBDs relationnels pour l'évaluation des requêtes , *l'évaluateur de requêtes* n'a que deux fonctions principales :

- *Optimisation de la requête XQuery globale* : cette fonction consiste dans la génération d'un plan d'exécution optimal.

- *La localisation des requêtes :* elle consiste dans la décomposition de la requête XQuery globale en sous requêtes XQuery, une requête par adaptateur. Les informations de localisation sont maintenues dans le cache du gestionnaire de vues. Il recomposera ensuite les résultats avant de les remettre à l'utilisateur final.

7. Architecture fonctionnelle

Pour implémenter l'approche citée ci dessus, l'architecture à trois niveaux est la plus appropriée. En effet, c'est une architecture qui permet de séparer clairement entre trois niveaux d'abstraction, à savoir (figure 6.24):

- *Le niveau présentation (service utilisateur) :* dans ce niveau, l'information est formatée afin d'être présentée à l'utilisateur. On retrouve dans cette couche , l'interface graphique client. Cette interface permet aux utilisateurs d'envoyer des requêtes sur la vue relationnelle globale.

- *Le niveau logique applicatif (service métier) :* cette couche fournit les interfaces de programmation nécessaires à l'implémentation des règles métiers du système. Cette couche permet d'implémenter le médiateur.

- *Le niveau données :* dans ce niveau on maintient les catalogues de données du système. A savoir l'ensemble des adaptateurs et donc les sources réelles des données.

Cette architecture permet d'intégrer des interfaces clientes existantes, en développant les vues relationnelles utilisateurs adéquates au niveau présentation.

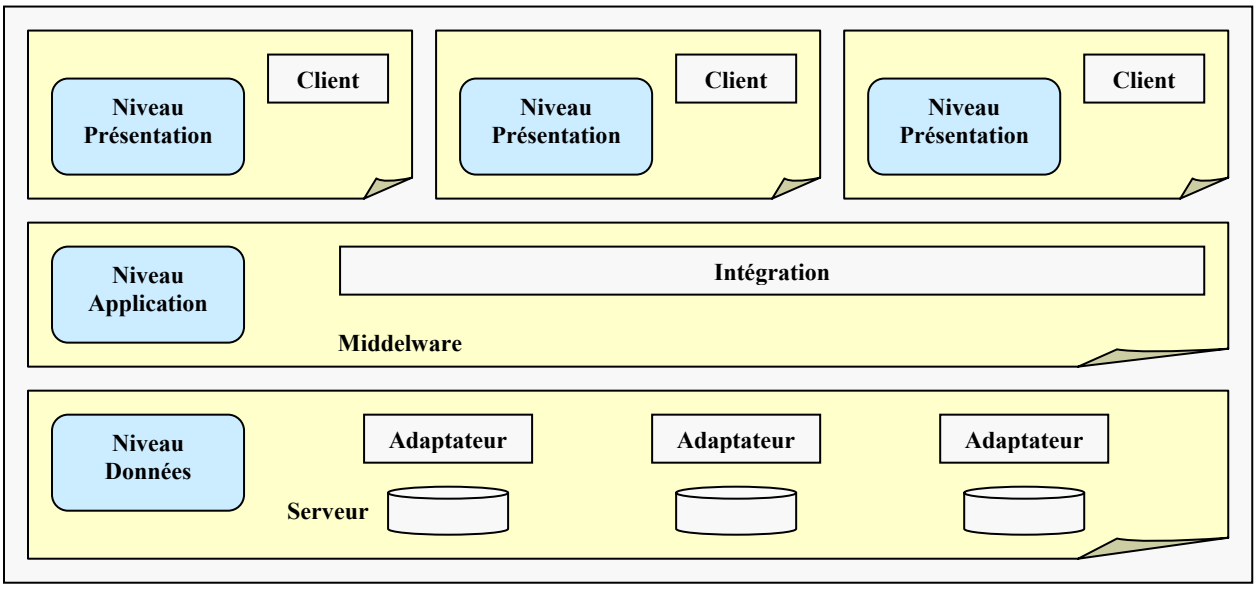

FIG. 6.24 Architecture à trois niveaux

Dans le chapitre (3), on a vue que les services web, manipulent des données sous forme XML, qu'ils permettent entre autre d'activer des services à distance sur des serveurs web distants. Couplés avec les protocoles de communication http et SOAP, ils présentent une infrastructure XML idéale , pour implémenter les différents composants de notre système , à savoir, le médiateur et les adaptateurs.

8. Exemple de motivation

L'architecture proposée dans ce chapitre s'inspire des architectures de fédération existantes , elle présente l'avantage d'utiliser des applications existantes en redéfinissant leur couche d'accès aux données selon le schéma relationnel global existant dans le médiateur. Notre approche s'inspire également d'un constat réel lors du développement du projet de fichier national des étudiants au profit du ministère de l'enseignement supérieur. En effet, les équipes de développement sont des équipes expérimentées en relationnel, capable d'exprimer les différentes règles métiers dans le modèle relationnel, quelque soit leur degré de complexité. La solution préconisée est celle du *warhousing* des données de l'année en cours.

Elargissons les besoins du système : il faut maintenant traiter des données qui s'étalent sur plusieurs années. Dans ce cas, la solution telle qu'elle a été mise en place, ne peut répondre à ce besoin. En effet, il s'avère impossible d'entreposer toutes ces quantités de données. De ce fait , on ressent les besoins utilisateurs suivants :

1. Accéder à ces données distribuées et hétérogènes (universités, résidences, ONOUs) en temps réel sans les entreposer au préalable au ministère .
2. Continuer à utiliser la même interface utilisateur. En effet c'est pas acceptable de former les utilisateurs à nouveau sur de nouvelles interfaces graphiques.

Du coté de l'équipe de développement , on ressent les besoins suivants :

1. Ne pas tout développer à nouveau, donc préserver les schémas globaux tels qu'ils étaient définis. Si extension nécessaire , utiliser l'expertise technique existant de l'équipe (relationnel /SQL).
2. Le besoin de continuer à utiliser une infrastructure solide ; à savoir les SGBDs relationnels qui sont matures et bien implémentés.

Une solution consiste dans l'utilisation de notre architecture à travers la démarche suivante :

1. L'équipe SQL exprime une solution SQL avec l'illusion « tout-relationnel » à travers l'élaboration du schéma global relationnel, et la définition des vues utilisateurs relationnelles adéquates.

2. La solution SQL sera traduite automatiquement par le système.

9. Conclusion

Nous avons présenté dans ce chapitre une architecture de médiation de base de données fédérée. Elle repose sur une approche de médiateur/adaptateurs qui est l'approche standard de développement des systèmes de base de données fédérées.

Le modèle de fédération commun entre les différentes sources hétérogènes que nous avons choisi est XML , et son langage de requête XQuery. En effet, c'est le format d'échange par excellence, puisque toutes les structures de données existantes peuvent être exportées vers le format XML.

Notre architecture présente la particularité de permettre aux applications pré existantes d'intégrer le système de fédération sans impact, ou avec impact minimum. Pour ce faire , elle se base sur les mécanismes suivants :

- *Une architecture de schémas :* capable d'assurer la traduction entre le schéma relationnel global relationnel, et le schéma interne modélisé en XML.

- *Un traducteur de requête* SQL –XQuery : qui permet de traduire les requêtes utilisateur , formulées en SQL , dans le langage XQuery.

- Un mécanisme d'exportation de données relationnelles en XML tout en permettant d'exécuter des requêtes SQL au niveau du moteur Relationnel.

Les Services web nous semble l'implémentation la plus adéquate aux différents composants décrits dans ce travail, puisqu'ils sont eux mêmes décrits en XML, qu'ils utilisent le format XML pour l'ensemble de leurs traitements. De plus, il permettent un déploiement via le Web.

Chapitre 7 : Conclusion

Dans cette thèse, nous nous somme intéressés au problème des bases de données fédérées basées sur XML. A l'apport de ce langage dans une architecture de médiation , plus particulièrement dans un contexte où le modèle relationnel est le modèle majoritaire dans le système.

Par le passé , les SGBDs fédérées ont tenté de répondre aux besoins de la distribution des données hétérogènes. Cependant, ils ont vite été abandonnés à peine expérimentés, parce que le modèle commun utilisé (relationnel ou objet) n'était pas suffisamment puissant pour répondre aux exigences de tels systèmes.

La seconde génération de solution consiste dans un ensemble d'outils collaborant ensemble pour offrir à l'utilisateur une vue homogène sur un système complètement disparate. Parmi lesquelles , seuls les entrepôt de données présentent une véritable alternative aux bases de données fédérées. En effet, ils permettent d'intégrer diverses sources hétérogènes dans une base unique. Cependant , la fraîcheur des données restent un point critique non résolu par cette approche.

Les données semi structurées, entre autre XML, qui ont émergé au départ du web, sont caractérisées par une structure irrégulière, dynamique ou inconnue ; néanmoins, elles permettent d'inclure la structure des données avec celles ci. Par conséquent, les données semi structurées représentent le modèle idéal, pour l'intégration des données hétérogènes. En effet, XML présente l'avantage d'être un modèle complet , permettant d'exprimer les relations, les attributs, l'héritage, le typage, l'hiérarchisation…etc. De plus, toutes les structures de données existantes peuvent être exportées facilement vers le format XML.

Une architecture de bases de données fédérées qui utilise XML comme modèle commun, doit s'appuyer sur un langage de requêtes expressif et puissant, pour permettre de manipuler les données . Parmi les langages de requêtes XML, XQuery est incontestablement le langage de requête le plus approprié pour les données XML car il est puissant, et permet d'exprimer des requêtes très complexes, à savoir des expressions de chemins, des jointures , des expressions conditionnelles…etc.

Basé sur XML, plusieurs architectures ont été proposées, soit dans des projets de recherches, soit dans des logiciels commerciaux. La plupart d'entre elles ont adopté l'approche de médiateur /adaptateurs de DARPA I3 ; où le médiateur s'occupe de la distribution des données ; alors que l'adaptateur s'occupe de l'hétérogénéité de celles ci.

Dans cette thèse nous avons présenté une architecture de médiation de base de données fédérée. Elle repose sur une approche de médiateur/adaptateurs. Elle est basée sur le couple XML \ XQuery. Elle présente la particularité de permettre aux applications pré existantes d'intégrer le système de fédération sans impact ou avec impact minimum. En donnant l'illusion « tout-relationnel » .

Le cadre général d'un système de médiation à base de XML à été décrit ; nous avons proposé entre autre une *architecture des schémas* ,capable d'assurer la traduction entre le

schéma global utilisateur relationnel, et le schéma interne modélisé en XML. Les mécanismes les plus importants de réécriture de requête dans ce contexte, ont été aussi décrits. Egalement une démarche de traduction des *requêtes* SQL vers XQuery à été décrite. Par choix de conception, notre architecture repose sur les moteurs relationnels sous jacents des sources locales. Pour ce faire, les données relationnelles locales sont exportées en XML. Enfin les grandes lignes d'une architecture fonctionnelle à été proposée.

Cependant, notre étude à permis de souligner les points :

- Difficulté de réécriture des requêtes selon l'ensemble des schémas relationnel ↔ XML.
- Nécessiter d'optimisation globale et locale des requêtes afin d'augmenter les performances du système.

Comme perspectives pouvant constituer une suite à notre travail on peut citer la mise en œuvre d'une théorie de vérification de la correction du passage SQL XQuery, pouvant préciser les requêtes traduisibles en XQuery.

D'autre part, le développement d'une plate forme basée sur les services web, permettant l'implémentation de cette architecture, peut s'avérer très prometteur, surtout quant à l'évolutivité du système du départ .

Bibliographie

[ABI 99] Serge Abiteboul , " On Views and XML"; Proceedings of the Eighteenth ACM SIGACT-SIGMOD-SIGART Symposium on Principles of Database Systems,1999, Philadelphia, Pennsylvania. ACM Press 1999.

[ABI 00] S.Abiteboul & P.Bumeman & D. Suciu, " Data on The Web : From Relations to semistructured Data and XML ", Morgan Kaufmann Publishers,2000.

[ABI 01] S. Abiteboul ; "A Dynamic Warehouse for the XML data of the Web"; Présnetation,Projet Xyleme. http://www.xyleme.com ,2001.

[ABI 02a] S. Abiteboul & Omar Benjelloun & Tova Milo ; " Web services and data integration "; Wise2002; Singapore, December 2002.

[ABI 02b] S.Abiteboul, "Web Services" ; INRIA-Futurs ; in Proc of VIII. Conference on Extending Database Technology (EDBT 2002) ; Prague; 2002.

[ABI 04] S. Abiteboul , "Management of semi structured Data" ; Support de cours; 2004.

[AVI 99] L.Avignon & T.Brethes & C.Devaux & P.Pezziardi ;"l'EAI :Intégration des Applications d ' E n t r e p r i s e" ; OCTO Technology Press. 1999

[BOT 03] P.Bothner , "What is XQuery? " XML Presse , 2003.

[BOU 94] N. Boudjlida et Y. Belhamissi , "Traitement de requêtes réparties : des modèles à leur implémentation ".In Actes des Journées Internationales des Sciences Informatiques, JISI'94, Tunis, Tunisie, Mai 1994.

[BOU 95] N. Boudjlida , " Intégration, Interopérabilité et Evolution des systèmes de données : application aux environnement logiciels ". thèse Habilité à diriger des recherches, Henry Poincaré –Nancy 1. 1995.

[BOU 96] N.Boudjlida ," Environnements Logiciels Intégrés et Bases de Données" , FIIA'96, Tunisie, Mars 1996 .

[BOU 02a] N. Boudjlida, " De la technologie Bases de données aux technologies Web " , MCSEAI Annaba, 2002.

[BOU 02b] N. Boudjlida, " Objets Structurés et Semi-Structurés: Perspective Bases de données " , Strasbourg, 2002.

[BOU 03] N.Boudjlida , "Bases de données Réparties " ; http://www.loria.fr/~nacer, 2003.

[BRA 03a] V.Braganholo & S.Davidson & C.Heuser ;"UXQuery: building updatable XML views over relational databases " .*Proceedings of the Brazilian Symposium on Databases;2003.*

[BRA 03b] V.P. Braganholo &S.B. Davidson&C. A. Heuser; "On the updatability of XML views over relational databases ";International Workshop on the Web and Databases (WebDB). San Diego, California. 2003,

[BUN 97] P. Buneman ;" Semistructured Data " ;Département of computer science, pennsylvannie . Proceedings of the sixteenth ACM SIGACT-SIGMOD-SIGART symposium on Principles of database systems 1997, Tucson, Arizona, United States.

[BUK 96] O.Bukhres & A. k. El magarmid , "Object oriented multidatatbase systèmes : a solution for advanced applications" ;Prentice Hall International, 1996

[CAU 01] Corine Cauvet & Camille Rosenthal-sabroux, "Ingénierie des systèmes d'information " ;Edition HERMES sciences Europe ,2001.

[CHA 94] S.Chawathe&H.Garcia-Molina&J.Hammer&K.Ireland&Y.Papakonstantinou &J.Ullman& J.Widom ."The TSIMMIS Project: Integration of Heterogeneous Information Sources ":In Proc. Of international processing Society of Japan Conf, Tokyo,Japan,October 1994.

[CHA 00] D.Chamberlin & J. Robie & D. Florescu , "Quilt: An XML Query Language for Heterogeneous Data Sources " Proceedings of WebDB 2000 Conference, in *Lecture Notes in Computer Science,* Springer-Verlag, 2000.

[CK 03] C.Knoblock , "Xquery Tutorial", University of Southern California.2003.

[CN 98] CNAM ; " DataWarhousing et DataMining " ;
Mémoire DEA , 1998 ;

[DAN 03a] Tuyêt Trâm DANG NGOC, " Fédération de données semi structurés avec XML", thèse de doctorat, Université de Versailles Saint-Quentin-en-Yvelines, 2003.

[DAN 03b] T.Dang-Ngoc & G.Gardarin, "Evaluating XQuery in a full-XML Mediation architecture" . Dans *les actes de publication de la conférence Bases de Données Avancées (BDA 2003)*, pages 283-302, Lyon, France, Octobre 2003.

[EDW 97] M.Edwards , "XML: Data the Way You Want It "; Microsoft Press,1997; xml.coverpages.org

[FAN 02] P.Fankhauser & P.Lehti , " XQuery by the Book - an Implementation Based on Rigid Formal Semantics" ; Procedeeing of XML Conference & Exposition 2002,Baltimore.

[FOU 01] X.Fournier-Morel ;"l'intégration d'information d'entreprises", SQLI Group, TRIBUNE *La chronique des NTIC*. 2001.

[FRN 98a] M.Frnandz & D.Florescu & j.kang & A.leavy & D.Suciu, " Web-site management : The strudel approche ".Bulletin of the IEEE Computer Society technical Comitee in data enginering 1998.

[FRN 98b] M.Frnandz & D.Florescu & j.kang & A.leavy & D.Suciu, "Overview of Strudel: a web site management système "; Networking and Information Systems Journal, Volume 1, p.115-140, 1998.

[FRN 00] M. Fernandez & W.Tan &D.Suciu; "SilkRoute: Trading between Relations and XML "; in Proc. 9th International World Wide Web Conference, Amsterdam, 2000 .

[GAR 91] G.Gardarin & P.Valduriez, "SGBD Avancés : bases de données objets, déductives, réparties " ; Edition Eyrolles, 1991.

[GAR 99a] G.Gardarin & Fei Sha &Tram. Dang Ngoc, "XML-based Components for Federating Multiple Heterogeneous Data Sources"; in 18th international Conf on Conceptual Modelig,volume 1728 of lectures Notes, Paris,1999.

[GAR 99b] G.Gardarin ; "Les DataWebHouses arrivent " ; octobre 1999.
Article paru sur www.gardarin.org

[GAR 02a] G.Gardarin & A.Mensch &A.Tomasic ," An Introduction to the e-XML Data Integration Suite " , Rapport technique, Mars 2002. www.e-xmlmedia.fr

[GAR 02b] G.Gardarin & *A. Mensch* & T.Dang Ngoc, *L. Smit*," Integrating Heterogeneous Data Sources with XML and Xquery"; 13th International Workshop on Database and Expert Systems Applications (DEXA'02); Aix-en-Provence, France. 2002.

[GAR 02c] G.Gardarin , "Intégrer vos données avec XML : e- xml integration suite"; BDA 2002 . 18èmes Journées Bases de Données Avancées ; Every, France 2002.

[GAR 03a] G.Gardarin, « XML : Des Bases de Données aux Services Web »
Editions Dunod ;2003 .

[GAR 03b] G.Gardarin &T.Dang-Ngoc; "Federating heterogeneous Data Sources With XML "; Dans *les actes de publication de IASTED International Conference on Information and Knowledge Sharing (IKS 2003)*, pages 193-198, Scottsdale, USA, Novembre 2003.

[GAR 04] G.Gardarin &T.Dang-Ngoc; " Mediating the Semantic Web ";Dans *un "keynote speech" des 4èmes journées d'Extraction et de Gestion des Connaissances (EGC 2004)*, à paraître (20 pages), Clermont-Ferrand, France, Janvier 2004.

[GIR 01] D.Girard & T.Crusson , "XML pour l'entreprise" ; Edition Improve copyright ; 2001.

[GRU 04] T. Grust & S.Sakr & J.Teubner, "Xquery on SQL Hosts " ; Université de Konstanz, Allmagne. Submitted, March 2004.

[KRE 01] H.Kreger ,"Web Services Conceptual Arechitecture : WSCA 1.0" ; IBM Systems Journal May 2001. IBM Softaware Group.

[LEE 01] D.lee & M.mani & F.Chiu & W.chiu; "Nesting based relational-to-XML schema translation"; ACM SIGMOD Int'l Workshop on the Web and Databases (WebDB), Santa Barbara, CA, May 2001.

[LEE 02a] D.Lee & M.Mani &F.Chiu &W.Chiu ; " NeT & CoT: Inferring XML Schemas from Relational World"; In 18th IEEE Int Conf on Data Engineering (ICDE2002), San Jose, CA, USA, Feburary 2002.

[LEE 02b] D.Lee & M.Mani&F.Chiu &W.Chiu ; " NeT & CoT: Translating Relational Schemas to XML Schemas using Semantic Constraints" ; In 11th ACM Int'l Conf. on Information and Knowledge Management (CIKM), McLean, VA, USA, November 2002.

[LES 98] A. Lesert; "XML-QL:A query Language For XML"; http://www.w3c.org/TR/Note-xml-ql/.

[LUD 99] B.Ludascher&Y. PapaKonstantinou & Pavel Velikhov & Voctor Vianu; "View definition and DTD inference for XML", Post-ICDT Workshop on Query Processing for Semistructured Data and Non-Standard Data Formats ***(SSD99)**, Jerusalem, January 1999.*

[MED 04] "Livre Blanc EAI" ; Edition MEDIADEV. 2004.

[MAN 00] I.Manolescu & D.Florescu & D.Kossmann & F.Xhumari & D.Olteanu: "Agora : living with XML and relationnel". Proceedings of 26th VLDB conférence , cairo, Egypt, 2000

[MAN 01] I.Manolescu & D. Florescu & D.Kossmann ,"answering XML Queries over hétérogenous data source "; Preceedings of the 27th VLDB conférence , Rome Italy 2001.

[MOL 95] H. Garcia-Molina, J. Hammer, K. Ireland, Y. Papakonstantinou, J. Ullman &J. Widom. *"Integrating and Accessing Heterogeneous Information Sources in TSIMMIS"*. In Proceedings of the AAAI

Symposium on Information Gathering, pp. 61-64, Stanford, California, March 1995.

[MOL 97] H. Garcia-Molina , Y. Papakonstantinou , D. Quass , A. Rajaraman , Y. Sagiv , J. Ullman , V. Vassalos , J. Widom. "The TSIMMIS approach to mediation: Data models and Languages". In Journal of Intelligent Information Systems, 1997.

[POP 02] L. Popa & Y. Velegrakis & R. J.Miller & M. A. Hernandez & R.Fagin , " Translating Web Data", Proc. 2002 *Very Large Databases Conference (VLBD'02).*Hong Kong, China ; 2002.

[ROB 03] J.Robie; "SQL/XML, XQuery, and Native XML Programming Languages"; Proceedings XML 2003 - Conference.Philadelphia; PA; 2003.

[ROB 04] J.Robie , "XQuery from the Experts - A Guide to the W3C XML Query Language." **;**Chapter 1, titled "XQuery: A Guided Tour,"; Ibm almaden research center; Addison-Wesley.2004.

[RYS 02] N.RYSER BOLOGNINI ; " Etude pour la création d'un entrepôt de données dans le cadre de l'assurance vie et transformation des données en informations utiles en vue d'une prise de décision" ; Mémoire Post grade en informatique et organisation ; Université de LAUSANNE ;2002.

[SHA 01] J.Shanmugasundaram & J.Kiernam & E.Shekita & C.Fan & J.Funderburk , "Querying XML Views of Relational DATA"; Preceedings of the 27th VLDB conférence , Rome Italy 2001.

[SIL 90] N.Silverio ; " Réalisation d'un compilateur " ; Edition Eyrolls 1990.

[SQL2] Norme SQL2 ANSI – BNF
www.netAktive.org

[STN 98] Stonebraker & Hellerstein; Readings in Database Systems, Morgan Kaufmann Publishers; 3rd edition ,1998.

[STO 01] Stohr E.& Nickerson J.V. " Intra Entreprise Integration :methods and direction" , Edition addison wesley,2001.

[STR 00] J. Strum , "Atelier XML ", Edition : Microsoft Press 2000.

[TES 00] O.TESTE ; "Modélisation et manipulation d'entrepôts de données complexes et historisées" ; thèse de doctorat ; Université Paul Sabatier, Toulouse; 2000.

[WG 93] Wiederhold G, *"Intelligent Integration of Information"*, ACM SIGMOD Conf. On Management of data, pp. 434-437, Washington D.C., USA, May 1993.

[XQ 03] W3C Working Draft ; " XQuery 1.0: An XML Query Language";12 November 2003.

Annexe

XQuery : Quelques propriétés

Les opérateurs XQuery

XQuery , comme la plupart des autres langages a des opérateurs arithmétiques, des opérateurs de comparaison et des opérateurs de séquences de nœuds. XQuery utilise l'opération appelée l*'extraction de valeur typée* pour correspondre les opérandes d'un opérateur XQuery. Prenons par exemple la requête suivante :

Doc(''books.xml'') /bib / book/ auteur[last ='stevens']

Dans cette requête l'opérateur = extrait la valeur typée de l'élément *last* qui est une valeur chaîne de caractère (string) et la compare à la chaîne de caractère *stevens.* L'extraction de la valeur typée est définie pour un élément unique. La forme générale de cette opération est appelée *atomisation* qui définit comment on extrait la valeur typée d'une séquence d'éléments. Par exemple pour la requête suivante : [XQuery Guide]

Avg(1 ,<e> 2 </e> , <e xsi: typr="xs:integer"> 3 </e>)

L'atomisation retourne la valeur typée de chaque élément dans la séquence qui est 2. pour les besoins de notre travail qui est de traduire à partir de SQL-XQuery , nous devons détailler comment le langage XQuery interprète les opérateurs :

a) Opérateurs Arithmétiques

XQuery supporte les opérateurs arithmétiques suivants : + , -, *, div (exécute une division sur n'importe quel type numérique), idiv (requiert un entier comme argument et retourne un résultat entier), mod. Si l'une des opérandes de l'opérateur arithmétique est un nœud on applique l'atomisation ; exemple :

2 + <int> {2 } </int> cette requête retourne 4 .

Si l'un des deux opérande est une séquence vide, le résultat est une séquence vide , donc dans l'exemple suivant , le résultat est la séquence vide :

2 + ()

Si par contre l'une des opérande n'est pas typée , elle est considérée comme de type *double.* Ce typage implicite est important surtout pour le traitement des document XML qui n'ont pas de XML-schema ,en conséquence ne sont pas typé (simple ou complexe).

b) Opérateurs de comparaisons

XQuery a les types d'opérateurs de comparaisons suivant : ***[ROB 04]***

- *Opérateurs de comparaison de valeur :* ils comparent deux valeurs atomiques. Si l'une des opérandes est un nœud , on utilise l'atomisation pour le transformer en valeur atomique ; par contre si l'une des opérandes n'est pas typée on utilise le type *chaîne de caractère.*
- *Opérateur de comparaison générale :* ils sont similaires aux opérateurs précédents puisqu'ils utilisent un quantificateur existentiel avec un opérateur de comparaison de valeur qui lui correspond. Le tableau suivant illustre ces deux types d'opérateurs : ***[ROB 04]***

Opérateurs de comparaison de valeur	**Opérateurs de comparaison générales**
Eq	=
Ne	!=
Lt	<
Le	<=
Gt	>
Ge	>=

Il existe deux différences majeurs entre ces deux types d'opérateurs :la première est que les opérateurs générale, après application de l'atomisation , permettent d'obtenir des séquences de valeurs atomiques alors que l'atomisation dans le cas des opérateurs de comparaison de valeurs retournent obligatoirement une valeur atomique unique. La seconde différence est que les opérateurs de comparaison de valeurs forcent l'utilisateur de typer les données d'une requête pour effectuer la comparaison ; alors que les opérateurs générales ne le font pas et s'ils comparent deux opérandes dans l'une des deux est non typée, ils procèdent comme suit :

- Si l'une l'autre valeur atomique de l'autre opérande est de type nimérique, alors le type requis est xs :double
- Si l'autre valeur atomique n'est pas typée aussi ; alors le type requis est xs :string
- Autrement , la valeur requise est de type dynamique qui correspond à l'autre valeur atomique.

Les opérateurs de comparaison de valeur traitent les séquences vides comme les valeurs nulles de SQL.

- *Opérateurs de comparaison de nœuds :* ***is*** et ***is not.*** ils déterminent si deux expressions sont évaluées au même nœud. Par exemple pour savoir si le livre le plus chère est celui qui a le plus de nombre d'auteurs et d'éditeurs.
- *Les opérateurs d'ordre :* XQuery définit deux opérateurs qui sont utilisés pour déterminer si un nœud est avant ou derrière un autre nœud dans l'ordre du document : l'opérateur **<< (a1 << a2** retourne vraie si a1 précède a2 dans l'ordre du document**)** et l'opérateur >> **(a1 >> a2** retourne vraie si a1 suit a2 dans l'ordre du document)

c) Opérateur de traitement des séquences de nœuds

XQuery fournit les opérateurs de traitement des séquences de nœuds suivants : *union (|)*, *intersect* et *except* . ils combinent deux séquences de nœuds et retournent une séquence de nœud dans le même ordre du document :

- *L'opérateur union :* il prend deux séquences de nœuds en entrée et retourne une séquence contenant tout les nœuds appartenant aux deux séquences.
- *L'opérateur intersect :* il prend deux séquences de nœuds comme opérandes et retourne une séquence contenant tout les nœuds qui apparaissent dans les deux opérandes en même temps.
- *L'opérateur except :* il prend deux séquences de nœuds comme opérandes et retourne une séquence contenant tout les nœuds qui apparaissent dans la première opérande et n'apparaissent pas dans la seconde.

Les fonctions XQuery

XQuery définit une librairie de fonctions complète, nous présenterons ici, les fonctions les plus répondues , à savoir :

- *Les Fonctions agrégats* Des fonctions d'agrégats sont définies dans XQuery comme *count, min, max.* l'exemple suivant illustre l'utilisation de la fonction agrégat *count.*
- *Les fonctions numériques :* roud(), floor() et ceiling().
- *Les fonctions de chaîne de caractères :* concat(), string-length() , starts-with(), end-with(), substring(), upper-case() , lower-case()
- *Des fonctions spécifiques au traitement de nœuds :* la fonction *not()* est utilisée dans des conditions booléennes , dans l'exemple suivant, la requête retourne les livres qui n'ont pas d'auteur dont le prénom est 'Stvens'

```
For $b in doc (''books.xml'')//book
Where not( some $a in $b/auteur satisfies $a/last=''stevens" )
Return $b
```

La fonction *empty()* quant à elle vérifie si une séquence est vide ; sa fonction opposée est la fonction *exists().* La requête suivante retourne les livres qui ont des auteurs, elle ne retournera donc pas le livre qui n'a que des éditeurs seulement :

```
For $b in doc(« books.xml'')//book
Where not ( empty(    $b/auteur)   )
Return $b
```

- *Des fonction d'accèes (accessor functions) :* ces fonctions accèdent à toute sorte d'informations associées avec les nœuds. La plus connues de ces fonctions est la fonction *string()* qui retourne la valeur chaîne de caractère d'un nœud , et la fonction *data()* qui retourne la valeur typée du nœud.

- *Des fonctions définies pas l'utilisateurs :* XQuery permet à l'utilisateur de définir ses propres fonctions afin qu'ils puissent simplifier des requêtes complexes, et de les réutiliser plusieurs fois, si besoin, dans la requêtes. Il permet aussi les fonctions récursives, qui sont très importante dans le contexte de document XML dont la structure est récursive.

Modules et définition de variables

XQuery permet de définir des variables dans le *prolog*. Apres sa déclaration, une telle variable est accessible à un point quelconque de la requête. Ainsi variables et fonctions peuvent êtres déclarés dans un module séparé qui peut être importé par n'importe quelle requête. Chaque module dans XQuery peut être soit le module *principal (main module)* qui contient le corps de la requête à évaluer, ou le module de *librairie* qui a la déclaration du module et non le corps de la requête.

Afin de permettre à des implémentations de XQuery d'être encapsulées dans des environnements tels que java ,c# ou base de données relationnelle. XQuery définit des fonctions et variable externes. Pour leur accéder, une requête doit les déclarer dans le prolog.