MINISTERE DE L'ENSEIGNEMENT SUPERIEUR ET DE LA RECHERCHE SCIENTIFIQUE

UNIVERSITE FERHAT ABBAS – SETIF
UFAS (ALGERIE)

MEMOIRE

Présenté à la Faculté des Sciences de l'Ingénieur
Département de l'Informatique
Pour l'obtention du diplôme de

MAGISTER

Option : Matériel et Logiciel

Par :
Mr: Samir FENANIR

THEME

DEVELOPPEMENT D'UNE SOLUTION AUTOUR D'XML POUR LA GESTION DES SYSTEMES D'INFORMATION HETEROGENES ET REPARTIS

Soutenu le : 13 mai 2008

Devant le jury composé de :

Président	: Dr M.TOUAHRIA	Maitre de conférence à l'UFA de SETIF
Encadreur	: Pr. M.MOSTEFAI	Professeur à l'UFA de SETIF
Examinateur	: Dr A.MOUSSAOUI	Maitre de conférence à l'UFA de SETIF
Examinateur	: Dr A.BENAOUDA	Chargé de cours à l'UFA de SETIF

DEDICACES

À

L’âme de Mon Père **AMMAR** (Que dieu l’installe dans les paradis),

Ma Mère **FATIMA** (Que dieu le protège),

Mes frères et sœurs,

Ma femme,

Ma Fille **HANANE**,

Je dédie ce mémoire.

Samir FENANIR

REMERCIEMENTS

En tout premier lieu, je tiens à exprimer ma plus vive reconnaissance à Monsieur **MOSTEFAI MOHAMED**, Professeur et Directeur à l'institut d'électrotechnique- université de Sétif, pour son accueil. Sa très grande culture scientifique n'a pas cessée de m'impressionner, de même que sa rigueur intellectuelle et son ouverture d'esprit. Je le remercie pour la confiance qu'il m'a porté en acceptant de diriger mes recherches.

Je tiens à remercier Monsieur **TOUAHRIA**, Maître de conférences à l'Institut d'informatique – université de Setif, pour m'avoir fait l'honneur de présider ce jury. Je suis particulièrement reconnaissant envers Monsieur A.**MOUSSAOUI**, Maître de conférences à l'institut d'informatique – université de Sétif et Monsieur **A.BENAOUDA**, Chargé de cours d'informatique – université de Sétif, d'avoir accepté la tâche d'examinateurs .

Enfin, je tiens à remercier tous mes amis qui, de près ou de loin, de part leur soutien moral ont contribué à l'aboutissement de ce projet.

Samir FENANIR

INTRODUCTION GENERALE

CHAPITRE 1: LES DONNÉES SEMI-STRUCTURÉES

CHAPITRE 2 : INTEGRATION DES SYSTEMES D'INFORMATION

CHAPITRE 3: UNE ARCHITECTURE DE MÉDIATION XML

CHAPITRE 4: VALIDATION

ANNEX

TABLE DE FIGURES

INTRODUCTION

Les besoins en matière d'information ont beaucoup évolués ces dernières années. Les nouvelles applications nécessitent des méthodologies flexibles et dynamiques pour partager des informations distribuées sur une multitude de sites. Alors qu'il y a vingt ans, les données étaient gérées sur un calculateur central, elles sont aujourd'hui disséminées dans l'entreprise dans des applications et des systèmes hétérogènes.

Certaines de ces informations sont enregistrées dans des bases de données traditionnelles et accessibles par des langages de requête très puissants, d'autres sont simplement stockées dans des systèmes de fichiers ou de simples tableurs, d'autres encore sont les données des pages Web.

La diversité de ces sources de données conduit à des modes de consultation qui peuvent être très différents. Ainsi, une base de données relationnelle sera interrogée par l'intermédiaire d'une requête SQL, une page Web sera consultée par une adresse Web (URL) particulière, et un tableur par une formule spécifique. Une telle variété de sources implique diverses façons de les interroger c'est-à-dire de formuler une requête, mais aussi plusieurs manières pour la source de présenter un résultat.

Plus précisément, il faut offrir un système de gestion intégrant des sources de données hétérogènes en assurant la transparence à la distribution et à l'hétérogénéité, ceci afin de faciliter l'accès pour l'utilisateur. La médiation de données est une réponse à ce besoin. Elle cherche à retrouver rapidement et facilement l'information disséminée sur un sujet donné.

La question subsidiaire est le choix du modèle global. Nous avons choisi XML comme un modèle global d'échange, c'est-à-dire comme un modèle pivot, et nous avons utilisé le langage XQuery comme un langage de requête commun pour interroger les sources de données.

L'évaluation de ces requêtes présente des aspects difficiles. En effet, une requête doit être divisée en sous-requêtes tenant compte de la localisation des sources et leurs capacités, celles-ci sont ensuite envoyées en parallèle sur les sites avec tous les problèmes d'ordonnancements et de synchronisations qui en découlent. Parmi les difficultés dégagées par la conception d'un composant de médiation on peut citer principalement:

- Comment intégrer des données de structures et de natures fondamentalement différentes ?
- Comment décomposer une requête faisant intervenir plusieurs sources en des requêtes spécifiques à ces sources, puis savoir recomposer le résultat ?
- Comment optimiser l'évaluation des requêtes dans un tel contexte distribué ?
- Que faire dans le cas des sources dont la possibilité d'interrogation est très limitée (ex. système de fichiers) ?

Dans ce contexte, le cadre de travail de ce mémoire se présente comme suit :

- Le premier chapitre Dresse un état de l'art sur les données semi-structurées, et leur langage de présentation XML.
- Le second chapitre expose les différentes approches d'intégration des données hétérogènes, en particulier les médiateurs.
- Le troisième chapitre présente une étude d'une architecture de médiation basée sur le langage XML, en distinguant les éléments blocs qui la composent.
- Le quatrième chapitre est consacré à la conception et à la réalisation du notre médiateur.
- Enfin, nous conclurons en résumant les différents concepts et mécanismes étudiés dans cette thèse, et en proposant de nouvelles perspectives de recherche.

1.1. INTRODUCTION

Les données du World Wide Web se caractérisent par des structures irrégulières dynamiques ou inconnues. Ces types de données sont communément appelées données semi-structurées. L'apparition du concept des données semi-structurées est considérée comme une révolution dans le monde des bases de données relationnelles et objets, les données semi-structurées ont pour objectif de permettre une meilleure représentation des entités du monde réel en intégrant le monde documentaire et le monde des bases de données relationnelles et objets.

En contrepartie, de telles données sont complexes à manipuler par des traitements automatiques. Les langages de requête, et les techniques d'optimisation et de stockage utilisés pour les données classiques ne s'appliquent plus. Il a fallu adapter de nouveaux concepts et algorithmes pour manipuler des objets semi-structurées.

1.2. NOTION DE DONNÉES SEMI-STRUCTURÉES

L'une des caractéristiques principales des bases de données relationnelles est la définition de schéma fixe auxquels les données sont ensuite obligées de se conformer. Il en va de même pour les bases de données objets. Cela simplifie les traitements informatiques (stockage, interrogations) et permet des accès sur critères très rapides. En revanche, une telle régularité ne permet pas de reproduire la pensée de l'utilisateur ou le monde réel. Ceci car les bases de données traditionnelles (structurées) ne peuvent pas travailler sur des données dont le schéma n'est pas fixé à priori. Cette opposition entraîne par la suite les problèmes bien connus suivants:

- *La structure de données peut évoluer:* cela nécessite alors la modification du schéma. Cette modification peut s'avérer complexe à réaliser.
- *Les données peuvent ne pas conformer exactement au schéma:* ceci nécessite le plus souvent de surdimensionner le nombre de colonnes et d'employer beaucoup de valeurs nulles.
- *Le même attribut peut avoir des types différents suivant les données:* cela nécessite le plus souvent l'emploi du type le plus englobant (souvent le type "chaîne de caractères"); ceci a pour conséquence de réduire la finesse de la description et les possibilités d'interrogation.
- *Une instance d'attribut peut être mono-valuée ou muti-valuée:* les SGBD traditionnelles traitent ces deux cas comme deux cas bien distincts, or il faudrait pouvoir gérer ces cas uniformément.
- *Les données peuvent être faiblement structurées (texte brut):* ceci entraîne l'utilisation de blocs de données brutes (BLOB) qui ne permettent pas des techniques très fines d'interrogation.

Les caractéristiques [Abi 98] des données semi-structurées sont décrites ci-dessous:

- *La structure est irrégulière:* une collection de données semi-structurées peut comporter des éléments hétérogènes de différents types pour représenter la même information. Un même attribut peut être mono-valué dans certaines instances et multi-

valués dans d'autres. Des informations supplémentaires (annotation, détails) peuvent apparaître à certains endroits. Enfin des éléments peuvent aussi manquer dans certaines instances.

- *La structure est partielle:* une partie des données peut être constituées d'information non structurées (images, textes bruts).
- *Le typage est irrégulier:* il n'y a pas de typage strict dû à l'hétérogénéité des données.
- *Le schéma peut être antérieur ou postérieur:* les SGBD traditionnelles sont basés sur l'hypothèse d'un schéma fixe défini avant toute insertion de données. Dans le cas de données semi-structurées, la notion de schéma est souvent postérieure à l'existence de données.
- *Le schéma évolue rapidement:* les sources de données semi-structurées sont habituellement dynamiques. Leurs données et leur organisation changent fréquemment. En conséquence, leurs schémas sont souvent mise à jour.

1.3. LE MÉTALANGAGE XML

XML (eXtended Markup Language) [Bra 98] est un métalangage permettant de générer d'autres langages tels que XHTML pour les pages WEB, WML pour les portables ou XMI pour représenter des modèles UML, XML est le standard d'échange proposé par le W3C. Il permet en effet d'échanger des informations marquées par balises décrivant la structure et la sémantique des contenus.

XML est un langage à base d'éléments, d'étiquettes, d'attributs et de valeurs. Les *balises* (*tag*) sont constituées *d'étiquettes* (label) permettant de marquer une information textuelle; les balises fonctionnent par paires: une balise d'ouverture, de la forme <nom>, et une balise de fermeture, de la forme </nom>, le composant logique compris entre les deux balises est appelé *valeur*. L'information textuelle marquée avec les balises qui l'encadrent constituent un *élément* XML. La valeur peut être vide, contenir de texte, d'autres éléments ou contenir un mélange des deux. L'élément de plus haut niveau englobant tous les autres et n'ayant pas de parents est appelé *élément racine*. Un élément peut contenir des informations additionnelles appelées *attributs*. 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="valeur". Un document XML est un ensemble d'éléments ainsi imbriqués.

Un document XML peut avoir deux qualifications, il peut être:

- ***bien formé*:** quand il respecte la syntaxe du langage XML définie par le W3C;
- ***valide*:** quand il est associé à une définition de type de document DTD et qu'il la respecte (nom des éléments, type, répétition et ordre d'apparition dans le document).

Un document bien formé est un document XML qui respecte certaines règles simples:

- Il existe un et un seul élément racine qui contient tous les autres éléments.
- Les balises sont correctement imbriquées: chaque balise ouvrante a une balise fermante associée et il n'y a pas de chevauchement.
- Le nom des balises est libre mais il contient au moins une lettre.
- Les attributs des balises, lorsqu'il existe, doivent comporter obligatoirement une valeur qui doit apparaître entre doubles apostrophes.

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

La figure 1.1. donne un exemple simple de document XML bien formé. La première ligne définit la version d'XML et le codage de caractères utilisés. L'élément racine, *document*, est constitué d'un titre, suivi de deux section incluant chacune un titre, la deuxième section incluant également une figure dont les caractéristiques sont données par les attributs *id* et *sysid*. Cet élément *figure* est un bon exemple d'élément vide, c'est-à-dire sans contenu, mais dont les caractéristiques apparaissent à travers les valeurs des attributs.

```
<?xml version="1.0" encoding="ISO-8859-1"?>
<document>
      <titre>Le Data Web: intégration des BD au Web</titre>
      <section>
            <titre>Introduction</titre>
      </section>
      <section>
            <titre>Emergence de XML</titre>
            <figure id="fig1" sysid="figure/fig1.cgm" />
      </section>
</document>
```

Figure 1. 1 - Exemple simple de document XML.

1.3.1. DTD et document valide

Un DTD (Document Type Definition) permet aux utilisateurs de définir leurs propres balisages, attributs et entités pour des documents de types SGML ou XML.

Un DTD se compose essentiellement de déclarations d'éléments et d'attributs associés. Une déclaration de la forme <!ELEMENT balise (…)> définit le modèle de contenu associé à l'élément ainsi balisé, c'est-à-dire les éléments composants et les attributs associés. Une déclaration de balisage peut être une déclaration de type d'élément, une déclaration de liste d'attributs, une déclaration d'entité ou une déclaration de notation.

Déclaration de type d'élément: Elle se signale par la chaîne <!ELEMENT. On déclare ensuite le nom d'élément suivi des spécifications de son contenu. Si l'élément est de type chaîne de caractères, on utilisera la chaîne #PCDATA. Dans le cas où l'élément comporte des sous-éléments, ceux-ci sont spécifiés. Il est ensuite possible de définir le nombre d'occurrences des sous-éléments avec 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.

Déclaration de liste d'attributs: On introduit une déclaration d'attributs par la chaîne <!ATTLIST et le nom de l'élément concerné. Vient ensuite l'ensemble des attributs de cet élément, suivi chacun d'une définition de type et d'un critère existentiel (#REQUIRED si

l'attribut est obligatoire, #FIXED si celui-ci est fixé, et #IMPLIED si l'attribut n'a pas de valeur par défaut et peut être déclaré ou non).

Déclaration d'entité: La chaîne <!ENTITY permet de définir une constante. La valeur de l'entité peut ensuite être appelée en précédant le nom de l'entité par le symbole '&'.

La figure 1.2 représente un exemple de DTD correspondant au document de la figure 1.1. On déclare que le document est composé d'un titre et de plusieurs sections (1 à N). L'élément *figure* est vide, mais possède deux attributs, *id* et *sysid*. L'élément *section* possède deux éléments imbriqués, l'un obligatoire (*titre*), l'autre est optionnel (*figure*). Le titre est du texte de type #PCDATA, c'est-à-dire devant être analysé plus avant. Notons que les attributs *id* et *sysid* sont aussi du texte mais cette fois de type #CDATA, c'est-à-dire ne devant pas être analysé plus avant. Enfin deux entités nom_bibliotheque et email sont définies et leurs valeurs sont respectivement "Bibliotheque Universitaire de Setif", "Biblio@uvs.dz".

```
<!--DTD for example 1 -->
<!ENTITY nom_ bibliotheque "Bibliotheque Universitaire de Setif">
<!ENTITY email "Biblio@uvs.dz">
<!ELEMENT document (titre, section+)>
<!ELEMENT figure EMPTY>
<!ATTLIST figure
      id CDATA #REQUIRED
      sysid CDATA #REQUIRED
>
<!ELEMENT section (titre, figure?)>
<!ELEMENT titre (#PCDATA)>
```

Figure 1. 2 - Exemple de DTD

Insuffisance des DTD : Les DTD définissent seulement la grammaire des balises, mais ne s'attaquent pas au problème de définir un schéma complet pour un document. Elles présentent de multiples limitations:

- Elles n'offrent pas de types de données simples autres que les types de données textuelles.
- Elles sont difficiles à interpréter et impliquent trop souvent un typage implicite avec de multiples conversions de types (par exemple des chaînes de caractères en réels).
- Elles sont difficiles à traduire en schéma objet et à générer à partir de schéma objet.
- Elles ne se définissent pas en XML, mais dans un langage spécifique.

Les schémas XML, que nous allons étudier dans la section suivante, visent à circonscrire ces limitations et, en conséquence, à remplacer à terme les DTD.

1.3.2. Les schémas XML

XML-Schema [tho 01][Bir 00] est un standard permettant de définir et de typer un document XML dans un formalisme proche de celui des bases de données. Il devient alors possible de définir des schémas de bases de données comme dans un modèle objet.

Les éléments et les attributs globaux sont créés par des déclarations qui apparaissent directement à l'intérieur de l'élément *schema*. Une fois déclaré, un élément ou un attribut global peut être référencé dans une ou plusieurs déclarations en utilisant l'attribut *ref*. Les éléments sont déclarés en utilisant le mot-clef *attribute*. Des types simples, tels que *string*, *token*, *int*, *decimal*, *time*, *date*, *ID*, *IDREF*, etc. sont prédéfinis dans XML-Schema. De nouveaux types simples peuvent être définis par dérivation des types simples déjà existants.

Les types complexes sont créés en utilisant l'élément *complexType* composé, dans la plupart des cas, d'une série de déclarations d'éléments et d'attributs et de référence d'éléments. Ces déclarations sont rassemblées dans des groupes modèles permettant de définir, un modèle de contenu constitué d'une *séquence* (ensemble ordonné), d'un choix exclusif *choice* ou d'un ensemble non-ordonné *all*.

Le nombre minimum (resp. maximum) de fois qu'un élément peut apparaître est déterminé dans sa déclaration par la valeur de l'attribut *minOccurs* (resp. *maxOccurs*). Cette valeur peut être un entier positif, ou encore le mot *unbounded* qui signifie qu'il n'y a pas de valeur limite. La valeur par défaut dans les deux cas (*minOccurs* et *maxOccurs*) est 1.
Exemple de schéma: Considérons le document représenté figure 1.3.

```
<?xml version="1.0" encoding='UTF-8"?>
<document version="1.0" date="16-03-01">
    <livre id = "1">
        <titre>xml</titre>
        <annee>2002</annee>
            <auteur>
              <nom>Nappa</nom>
              <adresse>aaaaaaa</adresse>
                  <etat>Californie</etat>
        </auteur>
     </livre>
    <livre id = "2">
        <titre>Base de donnees</titre>
        <annee>2000</annee>
          <auteur>
            <nom>Adiba</nom>
            <adresse>adadada</adresse>
                <pays>France</pays>
          </auteur>
    </livre>
 </document>
```

Figure 1. 3 - Exemple de document à schématiser

Pour écrire un schéma pour ce document, il faut tout d'abord ouvrir un élément xsd:schema puis suivre la structure et la définir en termes de types imbriqués. Le schéma étant lui-même un document XML. Nous définissons ensuite le type de l'élément racine

document, qui est bien sûr un type complexe. La liste des éléments enfants de *document* est une séquence composée du seul élément *livre* (*choice* ou *all* conviendraient tout aussi bien puisque la séquence comporte un élément unique), il s'agit ensuite de définir l'élément livre qui apparaît plusieurs fois. Celui-ci est lui-même composé d'une séquence d'éléments. Cette fois, l'élément *sequence* joue pleinement son rôle puisqu'un *livre* est une liste ordonnée des sous-éléments *titre*, *annee* et *auteur*. L'élément auteur est lui-même un type complexe, constitué d'une séquence d'élément *nom*, *adresse*, *pays* ou *etat*. Le choix entre *pays* et *etat* est exprimé par un constructeur *choice*. Il faut ensuite définir l'attribut obligatoire de *livre* nommé *id*. Enfin, après avoir fermé les balises ouvertes jusqu'à remonter au nœud *document*, il faut définir les attributs *version* et *date* de cet élément de type complexe. Il ne reste plus qu'à fermer le type complexe, l'élément et le schéma.Le schéma complet est donné figure 1.4.

```
<xsd:schema xmlns:xsd=http://www.w3.org/2000/10/XMLSchem>
<xsd:element name="document">
  <xsd:complexType>
    <xsd:sequence>
      <xsd:element name="livre" maxOccurs="unbounded">
        <xsd:complexType>
          <xsd:sequence>
            <xsd:element name="titre" type="xsd:string"/>
            <xsd:element name="annee" type="xsd:short" minOccurs="0"/>
            <xsd:element name="auteur">
              <xsd:complexType>
                <xsd:sequence>
                  <xsd:element name="nom" type="xsd:string">
                  <xsd:element name="adresse" type="xsd:string">
                  <xsd:choice>
                    <xsd:element name="etat" type="xsd:string">
                    <xsd:element name="pays" type="xsd:string">
                  </xsd:choice>
                </xsd:sequence>
              </xsd:complexeType>
            </xsd:element>
          </xsd:sequence>
        <xsd:attribute name=id" type="xsd:byte" use="required"/>
        </xsd:complexType>
      </xsd:element>
    </xsd:sequence>
    <xsd:attribute name="version" type="xsd:decimal" use="required"/>
    <xsd:attribute name="date" type="xsd:date" use="required"/>
  </xsd:complexType>
</xsd:element>
</xsd:schema>
```

Figure 1. 4 - Schéma pour le document de la figure 1.3.

1.3.3. Langages de requêtes pour données semi-structurées

L'interrogation de documents semi-structurés nécessite la définition d'un nouveau langage de requêtes. Ce qui est attendu d'un tel langage de requête [Abi 97]:

- Des opérateurs standards de requêtes sur base de données: les opérateurs habituels en base de données comme les projections, sélections, jointures, etc. doivent être définis;
- La possibilité de naviguer dans les données: le langage doit intégrer des notions de chemins;
- La recherche par motif: le langage doit permettre une recherche par mot-clef ou motif;
- La possibilité d'interrogation simultanée du schéma et de données: il faut pouvoir combiner des interrogations sur les éléments du schéma et des interrogations sur les données dans la même requête;
- La construction du résultat: il faut pouvoir spécifier la façon dont le résultat devra être présenté.

Plusieurs langages répandant plus ou moins à ces critères ont été proposés pour permettre l'interrogation de données semi-structurées.

1.3.3.1. Le langage LOREL

Le langage LOREL a été conçu à Stancford en 1996 pour interroger des graphes OEM à partir du langage OQL. La construction de base de LOREL est l'expression de chemin simple. Une expression de chemin simple est une séquence de balises séparées par des points décrivant un parcours dans le graphe OEM. Voici un exemple de requête utilisant de telles expressions:

(Q1): Renvoyer le nom de la personne dont la profession: "ingénieur"

```
SELECT personne.nom
WHERE personne.profession = "ingénieur"
```

Au-delà des expressions simples, LOREL permet l'utilisation d'expressions de chemins généralisées faisant intervenir certains motifs à la place de quelques étiquettes. Ces motifs ont des significations particulières comme un chemin optionnel (?), l'opérateur de répétition de kleen (*), une chaîne de caractère quelconque (%), une disjonction (|), etc.

(Q2): Renvoyer toutes les valeurs des composants des hôtels de villes ou de rues dont le Libellé contient la chaîne "Le Pont"

```
SELECT Hotel.#
FROM Répertoire.Hotel &H
WHERE &H.Adresse.(Ville | Rue) = "%Le Pont%"
```

1.3.3.2. XPath

XPath [Cla 99] est un langage de requête minimal. Une expression XPath est similaire à l'expression de chemin généralisée de LOREL, avec une syntaxe différente. Une expression XPath utilisée comme requête 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. Par exemple:

```
//livre/auteur/nom
```

Renvoie tous les noms d'auteur d'un livre, et

```
//livre [@id = "04242"]
```

Renvoie tous les livres dont l'attribut *id* vaut 04242

1.3.3.3. XML-QL

XML-QL proposé par AT&T en 1997 est l'un des langages de requêtes les plus intéressants. Ce langage est basé sur la reconnaissance de motif dans un graphe (pattern matching). Le langage XML-QL est déclaratif, intègre tous les opérateurs relationnels pour l'extraction, la conversion et la transformation de données XML. Il est également possible de formuler des requêtes sur des expressions de chemin. Le langage permet de plus la construction de document avec des requêtes imbriquées et les fonctions de skolems.

Une requête formulée dans ce langage comporte deux parties: une partie qui filtre le document XML par des variables, et une partie qui construit le résultat à l'aide des variables. Les données d'un document sont 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. La construction du résultat se fait en donnant le modèle du résultat final et en utilisant les variables de la partie condition ou des constantes.

Considérons le document XML "bibliographie.xml" se conformant au DTD:

```
<!ELEMENT bibliographie (livre *)>
<!ELEMENT livre (date, titre, auteur*)>
<!ELEMENT date (#PCDATA)>
<!ELEMENT titre (#PCDATA)>
<!ELEMENT auteur (nom, prenom, adresse)>
<!ELEMENT nom (#PCDATA)>
<!ELEMENT prenom (#PCDATA)>
<!ELEMENT adresse (#PCDATA)>
```

Figure 1. 5 - DTD du document "bibliographie.xml"

Exemples XML-QL

(Q1): Renvoyer la date et le titre des livres écrits par un auteur dénommé "Card". Le résultat devra être présenté avec des balises en anglais

```
WHERE
      <livre>
            <date> $d </date>
            <titre> $t </titre>
            <auteur>
                  <nom> card </nom>
            </auteur>
      </livre> IN "bibliographie.xml"
CONSTRUCT
      <book>
            <date> $d </date>
            <title> $t </title>
      </book>
```

Dans la requête (Q1), les variables $d et $t sont liées respectivement aux contenus de date et titre des éléments *livre* dont le nom de l'auteur est "card".

XML-QL permet d'exprimer des jointures. Celles-ci s'effectuent simplement par la réutilisation des variables dans des conditions. Les restrictions se font en utilisant la variable dans un prédicat. La requête (Q2) illustre un exemple de requête faisant intervenir des jointures et des restrictions.

(Q2): Renvoyer les emplois des auteurs des livres dont la date est postérieur à 1993 et dont le titre contient le mot "LINUX"

```
WHERE
     <livre>
           <date> $d </date> IN $d > 1993
           <titre> $t </titre> IN contains ($t, "LINUX")
           <auteur>
                 <nom> $n </nom>
                 <prenom> $p </prenom>
           </auteur>
      </livre> IN "bibliographie.xml"
      <personnel>
           <nom> $n </nom>
           <prenom> $p </prenom>
           <emploi> $e </emploi>
     </personnel> IN "personnels.xml"
CONSTRUCT
```

```
<auteur>
        <nom> $n </nom>
        <prenom> $p </prenom>
        <emploi> $e </emploi>
</auteur>
```

XML-QL permet des requêtes imbriquées dans la clause CONSTRUCT afin d'imposer des contraintes additionnelles aux données sélectionnées. La requête (Q3) récupère les variables $t et $d indépendamment et le résultat contiendra des combinaisons des éléments *titre* et *cadre*.

(Q3): Renvoyer les dates de publication des livres écrits par "Card" par titre.

```
WHERE
        <livre> $l </livre> IN "bibliographie.xml"
        <titre> $t </titre>
        <auteur>
                <nom> Card </nom>
        </auteur> IN $l
CONSTRUCT
        <resultat>
                <titre> $t </titre>
                WHERE
                        <date> $d </date> IN $b
                CONSTRUCT
                        <date> $d </date>
        </resultat>
```

En résumé, XML-QL est un langage de requête assez complet qui supporte toutes les opérations de l'algèbre relationnelle. Il est adapté aux documents XML par l'ajout de fonctionnalité d'expression de chemin.

1.3.3.4. XQL

C'est une variante du langage XPath proposée par Microsoft et d'autres constructeurs. Le langage est similaire à ceux vus ci-dessus, mais la syntaxe est différente. Au lieu d'utiliser des canevas XML, XQL propose d'étendre les URL pour interroger des collections de documents XML avec des expressions XPath. En particulier, les notations suivantes sont introduites, celles pour sélectionner les nœuds résultats allant au-delà des expressions de chemins XPath:

- / pour explorer le niveau suivant à partir de la racine ou de l'élément courant;
- // pour explorer tous les niveaux à partir de la racine ou de l'élément courant;
- * pour désigner un label quelconque;
- @ pour désigner un attribut (ex. @att);
- @* pour désigner un attribut de nom quelconque;
- […] pour définir des critères de filtrage;

- ? pour désigner un nœud résultat;
- ?? pour désigner la racine d'un sous-arbre résultat.

Exemples XQL:

(Q1): Retrouver tous les auteurs des livres de la bibliographie dont le nom de l'auteur contient la valeur "Card"

```
/bibliographie/livre/auteur??/nom [text () = "Card"]
```

Ou plus généralement:

(Q2): Retrouver tous les auteurs dont le nom est "Card"

```
//auteur??/nom [text () = "Card"]
```

XQL permet aussi de gérer un élément courant noté "." et de formuler des requêtes à partir de là, d'insérer des prédicats de comparaison, des méthodes, des expressions booléennes, de gérer des vecteurs, etc. il présente l'avantage d'étendre directement les notations des URL mais reste cependant loin des possibilités de XML-QL et XQuery, notamment en matière de restructuration de résultats.

1.3.3.5. XQuery

XQuery est le nouveau standard retenu par le W3C en 2001, il reprend les avantages de XPath, XML-QL et XQL. C'est un langage de type fonctionnel fortement typé basé sur les expressions de chemins.

1.3.3.5.1. Expression XPath

Pour adresser des éléments imbriqués dans les arbres XML, XQuery utilise XPath. Par exemple, la requête suivante porte sur un document et retourne un nom unique.

(Q1): Lister les auteurs des livres de la bibliographie

```
document ("bibliographie.xml") //livre/auteur/text()
```

Au contraire, la requête suivante porte sur une collection de documents XML et retourne une liste d'auteurs, les auteurs de tous les livres figurant dans la collection "biblio".

(Q2): Lister les auteurs des livres de la collection "biblio"

```
collection ("biblio") //livre/auteur/text()
```

1.3.3.5.2. Expression FLWR

Une expression FLWR est de la forme **for** … **let** … **where** … **return**. Les expressions FLWR sont similaires aux blocs **select** … **from** … **where** … **group by** de SQL, mais elles sont beaucoup plus puissantes. La forme générale plus précise d'une requête FLWR est la suivante:

```
for $var in foret [, $var in foret]*
let $var := sous arbre
where condition
return resultat
```

L'itération **for** permet de lier un tuple de variables à des arbres instances de forêts. Chaque instance de variable représente un arbre. Les forêts sont des collections d'arbres définies par des expressions XPath de la forme:

```
collection ("nom") / expression de chemin XPath
```

La clause **let** permet de collecter des arbres définis par des expressions XPath dans des variables. L'expression XPath peut être soit absolue, soit définie par rapport aux variables d'itération définies dans la clause **for**. La clause **let** est optionnelle, mais peut figurer en début de requête en place de la clause **for**.

La clause **where** permet de sélectionner les types de variables pertinents pour construire la réponse. La sélection se fait par une expression logique de prédicats élémentaires.

Enfin la clause **return** permet de construire les instances des documents résultat à partir des tuples de variables liées sélectionnées.

Exemple XQuery

(Q1): Renvoyer le nom de l'auteur et le titre des livres dont la date est postérieur à 1993 et dont le titre est "Linux Kernel 2.0"

```
for $p in collection("bibliographie") / LIVRES
where
      $p/date > 1993
      and
      contains($p/titre, "Linux Kernel 2.0")
return
    <livre>
      <auteur> $p/auteur/nom/text() </auteur>
      <titre> $p/titre/text() </titre>
    </livre>
```

1.3.3.5.3. Imbrication de requêtes

Il est possible d'imbriquer des requêtes au niveau du **for** (pour définir des variables sur des forêts calculées), au niveau du **where** (pour calculer des valeurs de prédicats), et au niveau de **return** (pour construire des documents imbriqués).

Exemples XQuery

(Q2): Trouver les livres dont le titre est "Linux Kernel 2.0", et qui sont écrit par un auteur dont le nom est "Card". Formater ensuite le résultat sous la forme donnée dans la clause **return**.

```
For $p in collection("personnel") / personne
For $l in collection("bibliographie") / livre
Where
      $l / titre = "Linux Kernel 2.0"
     and
       $l / auteur /nom = $p / nom
     and
       $p / nom = "Card"
return
      <auteur>
            <id>
                 <nom> $p/nom/text() </nom>
                 <prenom> $p/prenom/text() </prenom>
                </id>
                 <livre> $l/titre/text() </livre>
       </auteur>
```

La requête renverrait:

```
<auteur>
      <id>
          <nom>Card</nom>
          <prenom>Rémy</prenom>
       </id>
       <livre>Linux Kernel 2.0</livre>
</auteur>
```

(Q3): Trouver tous les éditeurs basés en France et pour chaque éditeur, afficher son nom, son adresse et tous les livres qu'ils éditent et dont le titre comporte le mot "Linux"

```
For $e in collection("editeur")/editeur
Where
  $e/pays = "France"
return
      <editeur>
          <nom> $e/nom/text() </nom>
```

```
        <adresse> $e/adresse/text() </adresse>
        <livres>
              for $l in collection("bibliographie")/livre
          where
             contains($l/titre, "Linux")
              and
                $l/editeur = $e/nom
              return
             <livre> $l/titre/text() </livre>
         </livres>
      </editeur>
```

La requête renverrait:

```
<editeur>
    <nom>Eyrolles</nom>
    <adresse>61 bd saint germain – 75005 Paris</adresse>
    <livres>
          <livre>Linux Kernel 2.0</livre>
          <livre>The Linux Process Manager</livre>
          <livre>Administrer un système Linux</livre>
          <livre>base de données</livre>
      </livres>
</editeur>
<editeur>
    <nom>OReilly France</nom>
    <adresse>18 rue séguier 75006 Paris</adresse>
    <livres>
          <livre>Le système Linux</livre>
          <livre>Les réseaux d'ordinateur</livre>
          <livre>Le système d'exploitation</livre>
          <livre>base de données et XML</livre>
      </livres>
</editeur>
```

1.3.3.5.4. Calcul d'agrégats

Des fonctions d'agrégation comme **count**, **min**, **max** sont définies en XQuery.

Exemple XQuery

(Q4): Donner le nombre de livre de la collection

```
let $l := collection("bibliographie") / LIVRES
return
      <nombreLivres> count ($l) </nobreLivres>
```

Cette requête est un agrégat simple qui profite de la puissance de mémorisation du **let**.

1.3.3.5.5. Recherche textuelle

Les recherches textuelles simples s'effectuent par le prédicat **contains** dans la clause **where**.

Exemple XQuery

(Q5): rechercher les livres contenant le mot "Linux" dans son titre et écrit après 1993

```
For $l in collection("bibliographie") / LIVRES
Where
      $l/date > 1993
    and
       contains($p/titre, "Linux")
return
      <livre>
            <titre> $p/titre/text() </titre>
      </livre>
```

1.3.4. Une algèbre pour XML

Dans le domaine des bases de données, il est classique de définir un ensemble d'opérateur de base permettant de traduire toute requête dans une séquence ou un arbre d'opérations élémentaires. Il s'agit de l'algèbre relationnelle pour SQL. Avec XML, c'est plus compliqué. Quoi qu'il en soit, nous aboutissons à la notion d'algèbre XQuery.

1.3.4.1 Algèbre XQuery

L'algèbre XQuery (Xquery Algebra) est un ensemble d'opérateurs élémentaires, chacun permettant de transformer une ou plusieurs collections d'arbres XML en une collection d'arbres, l'ensemble permettant de représenter des plans d'exécutions calculant la réponse à toutes requêtes XQuery.

XQuery est un langage fonctionnel, les opérateurs agissent sur des séquences d'arbres contenant de 0 à N arbres. La notation *$var* désigne une variable représentant à un instant donné un seul arbre. *Expr* désigne une expression pouvant être une constante, une séquence ou un template XML (document intégrant des requêtes). Une qualification notée *Qual* est une expression logique de comparaison entre expressions. Nous présentons dans le tableau ci-dessous (figure 1.6) les principaux opérateurs proposés.

Nom	Notation	Rôle
Assignation et construction	**let** $var := expr	Construction d'un élément XML et assignation à une variable $var.
Projection **XPROJECT**	$var/xpath	Extraction de séquences d'éléments ou d'attributs d'un arbre XML à l'aide d'une expression de chemin XPath.
Accès données **XSOURCE**	$var/xpath/data( )	Extraction de séquences de données d'un arbre XML à l'aide d'une expression XPath et de la fonction data( ).
Itération	**for** $var **in** seq **return** expr	Itération sur les éléments d'une séquence et construction d'une séquence résultat dérivée.
Sélection	**for** $var **in** seq **where** qual **return** cons	Itération avec sélection des arbres satisfaisant la qualification qual; une qualification est une expression logique de qualification simple de la forme : expr1 θ expr2
Jointure **XJOIN**	**for** $ var1 **in** seq1, $var2 **in** seq2 **where** qual **return** expr	Expressions d'itérations imbriquées permettant de joindre la séquence seq1 et la séquence seq2 selon la qualification qual. Le résultat est ordonné selon l'ordre des variables d'itération.
Tri	Expr1 **sortby** Expr2	Tri de la séquence Expr1 selon les données extraites par Expr2
Fonctions intégrés	**distinct-value** **unordered** **parent** **ref, deref** **index** **before, after** **avg, count, max, min, sum**	Elimination des doubles. Non prise en compte de l'ordre. Accès au nœud parent. Référencement et déférencement. Accès au rang d'un élément dans une séq Test comparé du rang de deux nœuds dans un document. Fonctions d'agrégats classiques.
Définition de fonction	**define function** nom ([type:var]*) **returns** collection	Définition d'une fonction paramétrée retournant une collection d'arbres XML

Figure 1. 6 - Les principaux opérateurs algébriques du XQuery

1.3.4.2. Le support des mises à jour avec XUPDATE

Pour manipuler les données, il faut aussi capable de les mettre à jour. XUpdate préconise une méthode déclarative pour insérer, supprimer et changer des noueds dans un document XML. XUpdate es donc basé sur XPath.l'approche générale consiste à utiliser un document XML de racine *modifications* pour contenir un ensemble de mises à jour possibles. Les

commandes s'appliquent sur un ou plusieurs noeuds contextuels sélectionnés par l'attribut *select* qui est valué par un XPath. Les commandes possibles sont les suivantes :

- **xupdate:insert-before :** insertion avant les nœuds contextuels ;
- **xupdate:insert-after :** insertion après les nœuds contextuels ;
- **xupdate:append :** insertion comme enfant des nœuds contextuels ;
- **xupdate:update :** mise à jour des contenus des nœuds contextuels ;
- **xupdate:remove :** suppresion des nœuds contextuels ;
- **xupdate:rename :** renemmage des nœuds contextuels ;
- **xupdate:variable :** définition d'une variable contenant une liste de nœud pouvant être réutilisée dans d'autres opérations d'insertion ou de mise à jour;
- **xupdate:if :** suppresion des nœuds contextuels ;

Les éléments xupdate :insert-before et xupdate :insert-after permettent respectivement d'insérer avant et après le nœud contextuel un nouveau nœud. Celui-ci construit par les éléments qui suivent. Ceux-ci sont choisis parmi :

- **xupdate:element :** création d'un élément ;
- **xupdate:attribute :** création d'un attribut ;
- **xupdate:text :** création d'un élément texte;
- **xupdate:processing-instruction :** création d'une instruction de traitement;
- **xupdate:comment :** création d'un commentaire ;

Par exemple, la commande :

```
<xupdate :modifications version "1.0 xmlnd :xu=http://www.xmldb.org/xupdate">
   <xupdate :insert-after select="/doc/livre[2]">
      < xupdate:element name="livre">
            < xupdate:attribute name "id">120</ xupdate:attribute>
            <titre>XML</titre>
            <edition>DUNOD</edition>
              <annee>2002</annee>
      </xupdate:element>
   </xupdate :insert-after>
</xupdate :modifications>
```

Insert le nœud *livre* d'identifiant *120*, de titre *XML*, de l'edition *DUNOD* et de l'année 2002 après le deuxième nœud d'une séquence de livres contenue dans l'élément *doc*.

1.3.5. Les API pour traiter les documents XML

XML n'est pas un langage de programmation, mais un métalangage permettant de représenter des données. Pour traiter ces données, il faut disposer d'un parseur, encore appelé analyseur.

Les parseurs permettent de réaliser l'interface entre une application et un document XML, produisant en sortie une structure de représentation interne (graphe d'objets ou flux

d'événements), vérifiant que le document est bien formé et optionnellement valide par rapport à une DTD ou un schéma.

Il existe deux types de parseurs: le parseur SAX (Simple API for XML) produisant un flux d'événements et le parseur DOM (Document Object Model) produisant un graphe d'objets en mémoire.

1.3.5.1. L'interface objet DOM

L'API DOM est basée sur une structure d'objets pour représenter un document balisé. Un parseur DOM génère donc un arbre d'objets reliés entre eux, un tel arbre est composé d'une racine *document*, de nœuds internes représentant les éléments ou les attributs, et de nœuds feuilles contenant les valeurs d'éléments ou d'attributs. Ceci conduit à des graphes volumineux, et il vaut mieux éviter d'utiliser DOM pour les documents importants car il est trop coûteux en mémoire.

Chaque type de nœud est muni d'une interface précisément définie. L'ensemble de ces interfaces compose l'API DOM. Toutes les interfaces héritent de *Node* qui permet essentiellement de parcourir et de modifier un graphe existant (retourne les enfants du nœud, retourne le parent du nœud, ajout d'un nouveau nœud, supprime un nœud, change la valeur d'un nœud, etc.).

1.3.5.2. L'interface événementielle SAX

SAX (Simple API for XML) offre un modèle plus simple d'événements déclenchés au fur et à mesure du parcours du document. Les types d'événements déclenchés sont début et fin de document, début et fin d'élément, un attribut à été reçu, un contenu texte à été trouvé, etc.

Ecrire une application SAX consiste à fournir l'implémentation de ces méthodes. Le parseur SAX appel les méthodes au fur et à mesure de l'analyse du flux XML.

SAX est une API légère et rapide. Elle ne construit pas d'image de document en mémoire mais permet les traitements à la volée.

1.3.6. Les feuilles de styles XSL

Les feuilles de style (Stylesheet) sont des fichiers associés aux documents XML contenant une séquence de commandes précisant les éléments de données à présenter et les mises en pages associées pour la publication de données sur un terminal (écran d'ordinateur, téléviseur, imprimante, fax, etc).

XSL est utilisé pour permettre de générer des présentations variées, par exemple en HTML, pour l'affichage sur écran, en WML pour les téléphones mobiles ou en RTF pour Word.

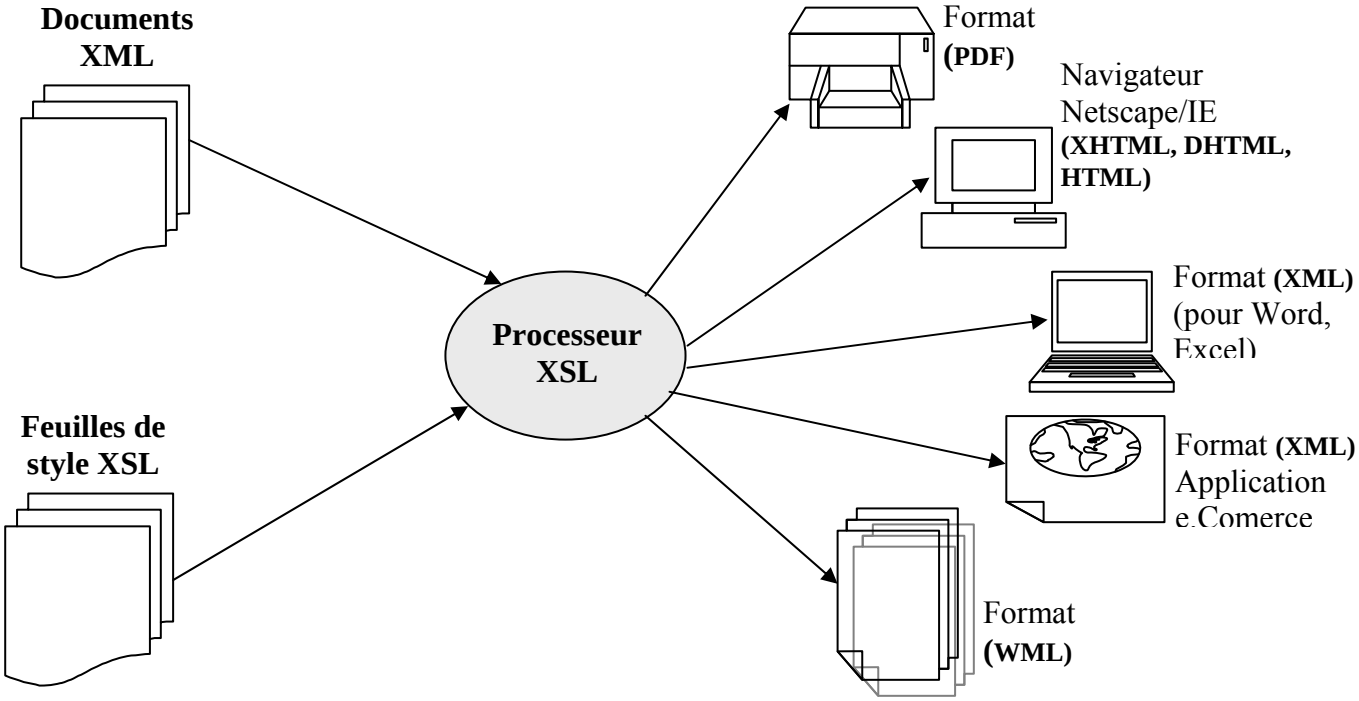


Figure 1. 7 - Génération de données publiables par un processeur XSL

Le couplage XML/XSL s'appuie sur un logiciel externe, que l'on appel processeur XSL, la tâche de ce dernier revient à produire des données publiables à partir du document XML en s'appuyant sur des feuilles de style écrites en XSL.

XSL est un langage de règles de production, chaque règle étant de la forme:

Condition ⟶ Action

La condition permet de sélectionner des nœuds dans l'arbre représentant le document alors que l'action spécifier les données à produire en résultat de l'application de la règle. Une feuille de style est donc une liste de règles, Chaque règle est exprimée en XML.

2.1. INTRODUCTION

Les données accessibles sur le réseau peuvent prendre différentes formes et ont chacune leurs spécificités propres. Il faut donc offrir un système de gestion intégrant des sources de données hétérogènes en assurant la transparence à la distribution et à l'hétérogénéité, ceci afin d'offrir à l'utilisateur une vue centralisée et uniforme des données.

2.2. LES DIMENSIONS FONDAMENTALES DANS L'INTEGRATION

Les systèmes d'informations peuvent être caractérisés [SUS 99] relativement aux dimensions : d'autonomie, d'hétérogénéité et de distribution.

2.2.1. L'autonomie

Dans notre travail, nous nous intéressons aux systèmes qui sont construits à partir de composants existants, ces composants sont disponibles et développés au sein des entreprises ou organisations (appelés : composants, application ou systèmes *patrimoines*). Les composants sont autonomes [VER 01] puisqu'ils peuvent s'exécuter en dehors du contexte du système qu'il intègrent (ils ont besoin que d'un ordinateur et d'un système d'exploitation). Alors, chaque entreprise dispose d'un système d'information qui assure sa bonne marche. Chaque système est conçu indépendamment par son propre concepteur, d'où, l'apparition de la problématique de l'hétérogénéité des systèmes d'information des différentes entreprises regroupés au tour du partenariat.

2.2.2. Hétérogénéité des sources de données

Nous pouvons communément décrire une source de données par [NGO 03] sa localisation, le type de données qu'elle gère, ses possibilités d'interrogation et le format des résultas.

- La localisation d'une source de données englobe tout aussi bien le référencement du site sur lequel se situe la source (URL, adresse IP+port, annuaire LDAP), que le protocole de communication utilisé (TCP/IP, IPX, Appletalk), les moyens d'accès à la base (ODBC, JDBC) ainsi que le support (pages Web, SGBD).
- Le type de données géré par une source peut être structuré (base de données relationnelles), semi-structuré (sources XML) ou non-structuré (images, multimédia, texte libre).
- Les possibilités d'interrogation sont aussi nombreuses: avec des langages de requêtes évolués et standardisés (SQL, OQL) ou propriétaires (LOREL), ou encore des recherches par motifs (moteur de recherche Web).
- Enfin, les format des résultats qui peuvent être formatées suivant divers standard (XML, HTML) ou encore accessibles par des structures de programmation variées.

2.2.3. Distribution des sources de données

La question de la distribution physique des sources de données est orthogonale à l'autonomie et à l'hétérogénéité des systèmes [VER 01]. Depuis déjà quelques années, il est

naturel de penser que les données et les traitements ne soient pas physiquement sur un même lieu ou machine mais, au contraire, répartis sur un réseau.

2.3. LES NIVEAUX D'INTEGRATION

On peut distinguer [GAR 06a] trois niveaux d'intégration (figure 2.1) : Intégration de données, Intégration d'applications, et enfin, l'intégration de plates formes. Dans ce chapitre, on va étudier ces différents niveaux d'intégration.

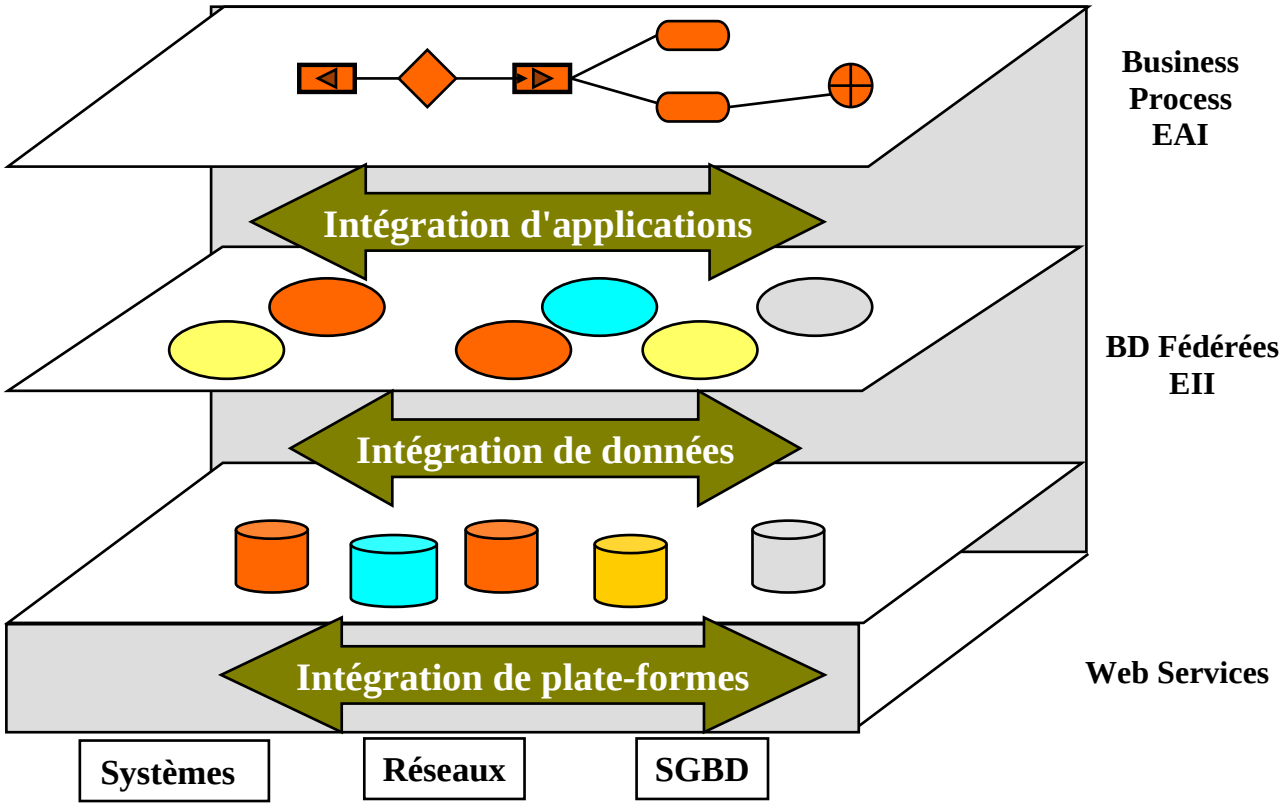


Figure 2. 1 - Les niveaux d'intégration

2.4. INTEGRATION DE DONNEES HETEROGENES

Il y a vingt ans, les données étaient gérées sur un calculateur central, elles sont aujourd'hui disséminées dans l'entreprise dans des applications et des systèmes hétérogènes. Aussi, accéder de manière intégrée à des données disséminées sur les différents sites du siège, des usines, des filiales, des partenaires, etc., est un besoin croissant. La fédération de données est une réponse à ce besoin.

2.4.1. Le besoin de fédération de données

Les systèmes d'informations [GAR 02] se sont développés depuis les années 70 en utilisant des modèles de données variés: modèle hiérarchique dans les années 70 avec COBOL, modèle relationnel dans les années 80, modèle objet intégré au relationnel dans les années 90. Dans les années 80, encore s'est développée la technologie des bases de données réparties. Une base de données répartie est un ensemble de bases de données gérées par des

sites différents et apparaissant à l'utilisateur comme une base unique. Elle est gérée par un SGBDR (Système de gestion de bases de données réparties).

Les limitations des bases de données réparties pour intégrer les données en entreprise ou sur le Web sont bien connues: nécessité de faire gérer chaque ensemble de données par un SGBD, forte dépendance créée entre les bases, modèle d'échange généralement relationnel et interrogation via SQL, difficulté pour intégrer des documents et des fichiers, recherche textuelle par mot-clés non supportée. Ceci a conduit les chercheurs, dès le début des années 90, à développer le concept de bases de données fédérées.

Une base de données fédérées est un ensemble de sources de données hétérogènes (fichiers structurés, bases de données, documents textuels…) accédées comme une seule via un modèle commun et un langage de requête commun. Alors que les bases de données réparties intègrent plusieurs bases souvent homogènes et sont gérées par un SGBD répartie, les bases de données fédérées sont gérées par un système de médiation de données.

2.4.2. Exemple de bases fédérées

La figure 2.2 représente un réseau typique des besoins actuels des entreprises. Il existe en effet souvent des bases de données traditionnelles (modèle relationnel ou objet), stockant des données de gestion de l'entreprise. Il peut aussi exister des bases de données techniques décrivant les produits fabriqués et leurs composants, des documents textuels contenant par exemple les manuels d'opérations, des données géographiques matérialisant la localisation des usines et des clients, etc.

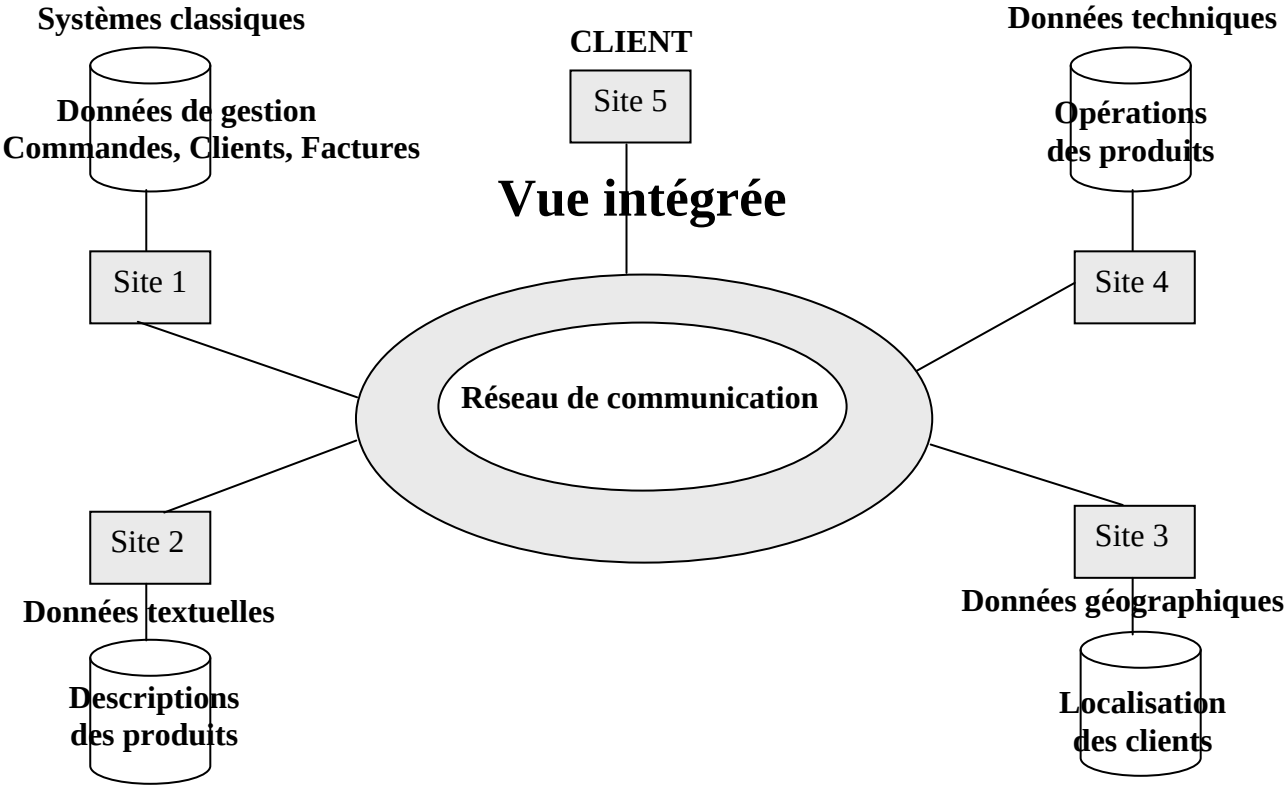


Figure 2. 2 - Exemple de bases de données fédérées

Le problème qui se pose est de faire intégrer toutes ces sources hétérogènes. Donc, on doit fournir une vue de données intégrée et un modèle d'accès unifié, afin d'obtenir une véritable base de données fédérée.

2.4.3. Modèle global d'échange XML

L'intégration point à point a été historiquement une approche pour combiner des données. Soit deux bases de données à intégrer, A et B. dans sa forme la plus simple, l'intégration point à point consiste à réaliser une passerelle de B vers A, permettant d'accéder aux données de B depuis A. Si A et B sont dans des modèles différents (par exemple relationnel et hiérarchique), il s'agit de convertir les données hiérarchiques en relationnelles, ce qui n'est déjà pas si simple. Pire, supposons que l'on veille, depuis M sources de données, accéder à N autres sources : il faudra alors réaliser M × N traducteurs (figure 2.3). Outre cet aspect quantitatif en nombre de traducteurs, l'intégration point à point de données dans un système d'information est coûteuse et difficile à maintenir. En effet, dès q'une source change, il faut modifier tous les traducteurs associés. Pire encore, lorsqu'une nouvelle source est ajoutée, il faut développer des traducteurs avec toutes les autres.

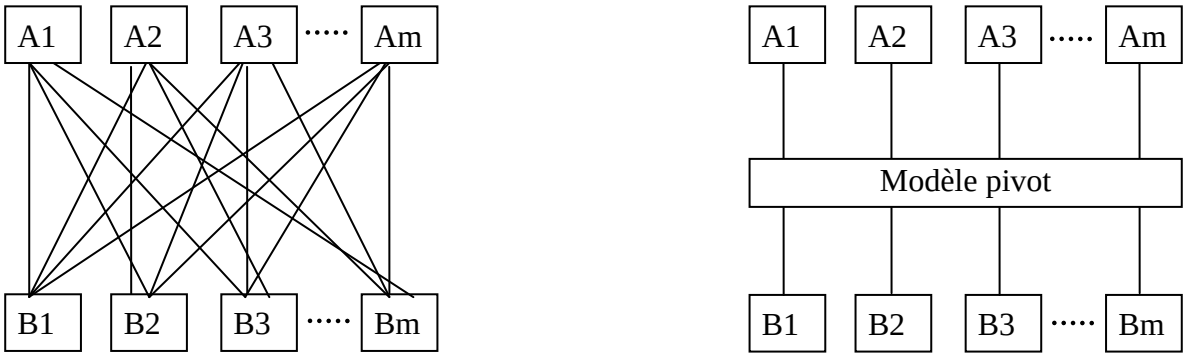


Figure 2. 3 - Intégration par modèle global versus modèle point à point.

L'intégration par modèle global est une approche beaucoup plus rationnelle et économique. Il s'agit de choisir un modèle d'échange pivot, dans lequel toute donnée est traduite avant d'être intégrée à d'autre source. Supposons que l'on veuille depuis M sources de données échanger avec N autres sources : il faudra alors réaliser seulement M + N traducteurs (figure 2.3). Pour ajouter une nouvelle source, il suffira d'un traducteur depuis cette source vers le modèle pivot. La maintenance d'une source n'impacte que le traducteur associé. Indiscutablement, passer par un modèle pivot pour intégrer des données hétérogène est une approche rationnelle et économique à l'intégration de données.

La plupart des systèmes de médiation ont aujourd'hui choisi XML comme modèle global d'échange, c'est-à-dire comme modèle pivot. Ce choix résulte des relatifs échecs des générations de systèmes basés sur un modèle global relationnel ou objet. Une raison plus positive est que XML est un modèle d'échange complet, qui intègre tous les concepts des modèles connus. En effet, Avec les schémas, le typage des données élémentaires est très complet. De plus, il est possible de créer des types complexes avec les constructeurs

séquence, choix et tas. Enfin, il est relativement facile de générer du XML depuis des formats divers : tables relationnelles, fichiers hiérarchiques COBOL par exemple, documents HTML ou RTF. XML paraît mieux adapté que le relationnel pour fédérer des données. En effet, le relationnel est un modèle pauvre en structures et sémantique. Par exemple, traduire des articles de fichiers COBOL en lignes relationnelles n'est pas simple car les articles sont multiniveaux en COBOL. La normalisation conduit à plusieurs lignes de tables différentes. Avec XML, c'est direct : on traduit un arbre en un arbre. Donc, XML est un modèle idéal comme modèle pivot dans un SGBD fédéré.

2.4.4. Techniques d'intégration de données hétérogènes

Les technologies de fédération de données vont dans deux directions : (i) bases de données virtuelles composées à partir de sources hétérogènes ; (ii) plate forme d'intégration ouverte basée sur un entrepôt de données.

La première est connue sous le nom de base de données hétérogènes intégrées ou encore base de données fédérées. Les bases de données fédérées permettent d'accéder sur demande par des requêtes. Elles constituent donc des vues intégrées virtuelles des données hétérogènes.

La seconde technique s'articule autour d'un entrepôt de données (*datawarehouse*). Un entrepôt sert comme système de stockage unifié pour des données intégrées extraites ou copiées depuis différentes sources de données. Dans ce cas la base de données fédérée est matérialisée à l'aide d'outils d'extraction.

Quoi qu'il en soit, l'intégration virtuelle de sources de données hétérogènes par un système de médiation présente de nombreux avantages, parmi lesquels :

- La possibilité de reprise des sources existantes dans un environnement intégré plus avancé par une interface de requête (API XQuery par exemple).
- La possibilité de prendre en compte des données géographiquement dispersées et leur diversité sémantique.
- Une plus grande flexibilité afin d'assurer le partage des données hétérogènes et réparties entre différents systèmes de médiations et systèmes locaux.
- Des performances espérées meilleurs du fait de la gestion de caches des bases intégrées au niveau du système de médiation.
- Une intégration des réseaux Internet ou intranet cachés par le système de médiation.

Cependant, l'intégration de sources hétérogènes, présente quelques difficultés majeures :

- La complexité de faire collaborer des modèles de données et des langages de requêtes différents.
- Le manque d'expérience notamment dans l'intégration des données hétérogènes sur le Web.
- La distribution du contrôle de données entre plusieurs sites, ce qui pose en général des problèmes de cohérence non triviaux lors des mises à jour généralement mal supportées.

- La difficulté de changement, par exemple pour intégrer un nouveau type de source de données, qui peut cependant être moindre qu'avec un système centralisé nécessitant une migration.

Il faut aussi dire que avant l'avènement de XML les systèmes fédérés se sont généralement appuyés sur le modèle relationnel. L'avènement de XML comme modèle d'échange, notamment sur le Web, devrait changer la donne, comme nous le verrons ci-dessous.

2.4.5 Architecture GAV et LAV

On peut classifier [HAL 00] suivant la relation entre les schémas des sources locales par rapport au schéma unifié global (figure 2.4) :

- GAV (Global As View) : schéma global défini comme une vue intégrante sur schémas locaux. C'est une approche ascendante depuit les sources vers le médiateur.
- LAV (Local As View) : chaque source locale est définie comme une vue locale du schéma global. Elle est descendante depuis le médiateur vers les sources.

Dans l'approche GAV, la transformation d'une requête sur le schéma global en requête sur le schéma local est une simple opération faite par le gestionnaire de vue. Dans le cas d'une approche LAV, la requête sur le schéma global doit être reformulée suivant les schémas des sources locales. D'un autre coté, dans une architecture GAV, une modification sur l'ensemble des sources locales ou sur leurs schémas entraîne une reconsidération complète du schéma global, dans l'architecture LAV, chaque source est spécifiée en mettant à jour la vue locale, de plus, si les données des sources locales n'on pas le même format relationnelle, semi-structurées), il est difficile de définir le schéma global comme vue des sources de différent formats.

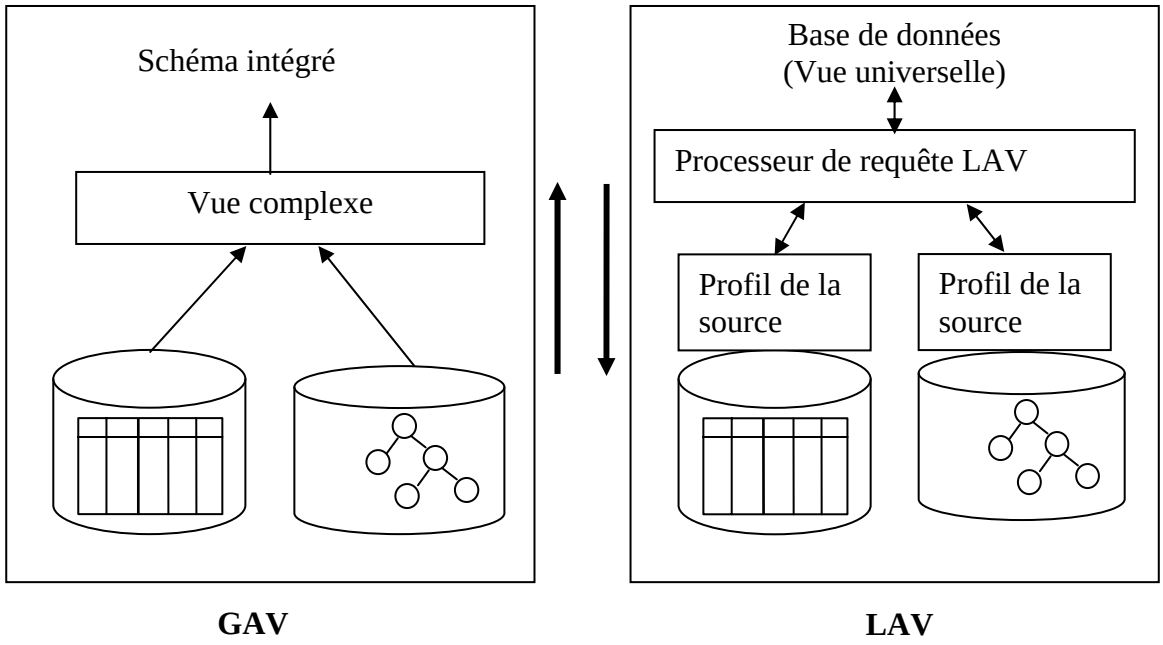


Figure 2. 4 - Architecture GAV et LAV

En utilisant une approche LAV, chaque source peut être décrite séparément par un mécanisme de vue spécifique à son format. Les architectures les plus souvent utilisées sont des architectures GAV (TSIMMIS, GARLIC, DISCO, YAT, XPERANTO). L'approche LAV est très peu utilisée (INFORMATION MANIFOLD, AGORA)

2.4.6. Principes généraux de la médiation

Le système de médiation de données (Data Mediation System) est un ensemble de modules logiciels permettant un accès unifié à des sources de données hétérogènes sur le Web, en Internet ou intranet.

Afin que l'utilisateur ait l'illusion d'un accès à une seule et unique base de données virtuelle, ce système doive posséder un langage de requête unique et retourner à l'utilisateur un résultat sous un format unique.

2.4.7. Problèmes de la médiation

L'optimisation des requêtes dans des bases de données hétérogènes conduit à de nombreux problèmes [NGO 03] qui peuvent se classer en deux catégories: l'intégration des données hétérogènes et l'évaluation de la requête.

2.4.7.1. Les anomalies dans l'intégration des données hétérogènes

Des anomalies fréquentes dans l'intégration des données hétérogène peuvent survenir:

- Redondance des données: ce type d'anomalie survient lorsqu'une même donnée est copiée sur différents sites.
- Correspondance des noms: on en distingue deux sortes:
  - Même nom d'entité mais significations différentes: par exemple, une même entité désignant une personne sera appelée "personne" sur un site, "p" sur un autre site, et "person" sur un autre;
  - Nom différents mais significations identiques: par exemple sur un site, "nom" désignera l'entité correspondant au nom de famille d'une personne, et l'entité correspondant au nom d'une société sur un autre site;
- Conflit de divergence schématique: ce type de conflit survient lorsqu'un même type de données est modélisé différemment. Par exemple, l'adresse d'une personne peut être représentée par une chaîne de caractère sur un site, et par un élément complexe comportant les attributs numero, rue, ville sur un autre.

L'existence de ces anomalies pose alors les problèmes suivant aux outils d'intégration:
Comment guider l'utilisateur dans la formulation d'une requête valide? C'est-à-dire, comment lui présenté les métadonnées afin qu'il puisse formuler une requête valide?
Comment présenter efficacement les métadonnées afin d'évaluer une requête et identifier les sources nécessaires à son exécution?

2.4.7.2. Evaluation de requête

Une requête formulée au médiateur est posée indépendamment de la localisation des différentes données intervenant pour calculer le résultat. Cela introduit trois difficultés:

La décomposition d'une requête: il s'agit à partir d'une requête posée, de localiser les données intervenant dans sa résolution, de produire des sous-requêtes spécifiques à chacune des sources et d'ordonner ces sous-requêtes.
La recomposition des résultats: une fois les sous-requêtes soumises à chacun des sources, il s'agit de savoir recomposer les différents résultats entre eux.
Au niveau de l'optimisation: le médiateur a rarement une vision sur la façon dont sont traitées les sous-requêtes au niveau des sources (placement des données, type de stockage, indexation, stratégie d'évaluation). De plus la distribution des données sur des sources disjointes ne permet pas d'utiliser directement les algorithmes d'optimisation employés dans le cas d'un SGBD centralisé.

2.4.8. Objectifs des médiateurs de données

Les objectifs des systèmes de médiation [GAR 02] se résument aux aspects suivants:

2.4.8.1. Accès intégré depuis un serveur d'application

Un utilisateur peut demander l'exécution d'une requête de recherche distante à un serveur d'application, peut par exemple être exprimée en SQL ou en XQuery avec XML. Les programmes d'applications doivent accéder au système de médiation par une API standard ou en utilisant les services Web.

- Avec SQL: JDBC est d'usage dans le cadre d'une architecture J2EE, et ADO SQL sont utilisés avec Microsoft Windows.
- Avec XQuery: XML/DBC est une API permettant d'émettre les requêtes XQuery et de récupérer les collections de documents résultant.

Un système de médiation doit permettre les recherches simultanées de plusieurs clients aux données, et il supporte encore les mises à jour. Pour assurer l'objectif de cohérence des mises à jour, le système de médiation doit fournir des mécanismes de contrôle de concurrence adaptés, garantissant que l'effet de l'exécution simultanée des transactions est le même que celui d'une exécution séquentielle.

2.4.8.2. Transparence à la localisation des données

Un des objectifs des systèmes de médiation est de permettre d'écrire des programmes d'application sans connaître la localisation physique des données. Dans ce but, les noms des objets doivent être indépendants de leurs localisations.

Les avantages de la transparence à la localisation sont tout d'abord de simplifier la vue utilisateur et l'écriture des requêtes, mais surtout d'introduire la possibilité de déplacer les objets sans modifier les requêtes.

2.4.8.3. Performance et passage à l'échelle

L'objectif des systèmes de médiation est de délivrer l'information en temps réel et à jour, et ils doivent idéalement pouvoir supporter un grand nombre d'utilisateurs simultanés en garantissant de bons temps de réponse. Une réponse à la performance est l'introduction de cache de données au niveau du système de médiation.

Une cache de données (Data cache) est un espace mémoire permettant de conserver les données les plus fréquemment accédées au sein d'un système. Elle peut être utilisés pour garder des index de localisation des données permettant un accès direct. Elle permet aussi de gérer des copies totales ou partielles, de se replier sur une copie lorsqu'une autre est indisponible (site en panne), et même de mettre à jour de manière indépendant des copies.

2.4.8.4. Autonomie locale

Un des objectifs importants des systèmes de médiation est l'autonomie locale, qui vise à garder une administration locale séparée et indépendante pour chaque serveur participant à la base fédérée. Ainsi, les reprise après panne doivent être accomplies localement et ne pas impacter les autres sites. Les mises à niveau de logiciel sur un site doivent être possibles sans impacter les autres sites. Comme chaque source collabore avec le système de médiation, les modifications des métadonnées locales (schémas locaux) doivent être prises en compte automatiquement par le système de médiation.

2.4.8.5. Intégration syntaxique des données

Un système de médiation doit être capable d'unifier les modèles de données supportés par les différentes sources participantes. Par exemple, on voudra créer une base unifiée en intégrant une base relationnelle, une base objet, un référentiel XML et un annuaire LDAP. Pour unifier les modèles, il s'agit de réaliser des transformations de structures de données à un modèle pivot. XML est un modèle pivot (global) idéal pour réaliser l'intégration syntaxique des modèles actuellement utilisés dans les entreprises.

Au-delà des modèles, il faut aussi intégrer les langages de requêtes. L'unification s'effectue par utilisation d'un langage de requête pivot associé au modèle pivot. XQuery avec XML est plus approprié pour des données complexes.

2.4.8.6. Intégration sémantique des données

Au-delà de la possibilité d'intégration syntaxique des données, le problème de l'intégration sémantique des données se pose. En effet, celles-ci ont souvent été conçues indépendamment si bien qu'une même donnée peut porter un nom différent et avoir une représentation différente. À l'opposé, deux données de même nom peuvent être sémantiquement différentes, il faut donc être capable de rapprocher des schémas, d'isoler les structures identiques, de reconnaître les données différentes et d'intégrer l'ensemble des schémas en un schéma unique.

2.4.9. Architecture de référence d'un médiateur

Les architectures des bases de données fédérées ont progressivement convergé vers une vue unifiée proposée par DARPA du groupe I3 (Information Integration Interoperability) en 1995. Cette architecture est représentée par la figure 2.5.

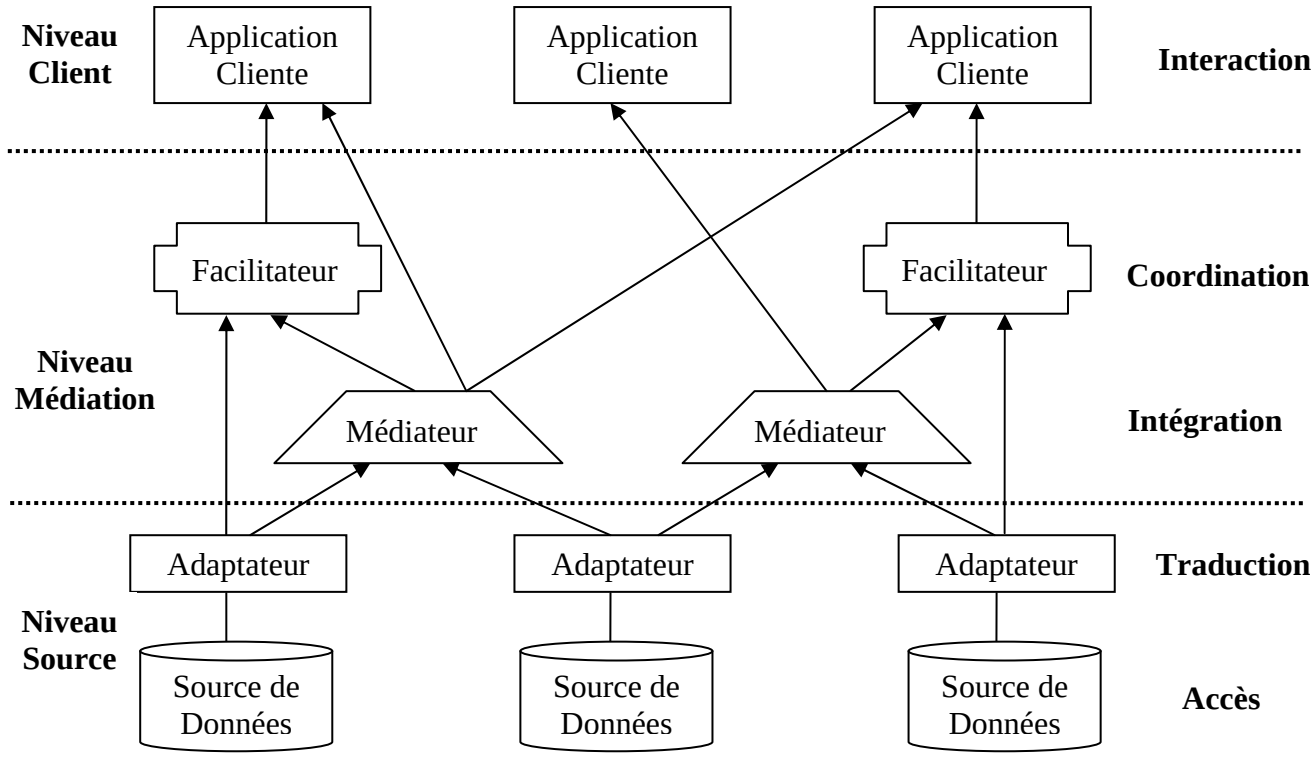


Figure 2. 5 - Architecture de référence d'un système de médiation

Cette architecture de référence se compose de trois niveaux:

1. **Le niveau source:** parfois appelé niveau local, il comporte:

- les différentes sources de données qui peuvent être très différentes (SGBD relationnelle, SGBD objets, système de fichiers, pages Web, Moteurs de recherche);
- Un adaptateur (wrapper) accepte une requête donnée dans le langage pivot du médiateur, la transcrit dans le langage natif de la source et exécute la requête. Ensuite, il récupère les résultats de la requête et les traduire en modèle pivot du médiateur et les renvoi à celui-ci.

2. **Le niveau médiation:** comporte:

- Des médiateurs permettant de décomposer la requête issue de l'application cliente en sous requête locale, et de récupérer les résultats de chaque source de données (concernée par cette requête) et d'assurer l'intégration des données pour composer la réponse aux clients;
- Des facilitateurs permettant de composer les réponses aux requêtes dans le format souhaité par le client. Typiquement, les documents XML pourront être transformés en

objets (cas d'un facilitateur objet) ou en HTML par des feuilles de style XSL (cas d'un facilitateur HTML).

3. **Le niveau client:** comporte les applications clientes (navigateurs, programmes d'application, interfaces graphiques), permettant de transmette des requêtes vers le niveau médiation et de présenter les résultats aux applications.

La communication entre le médiateur et les sources de données se fait via le protocole SOAP qu'on va étudier à la fin de ce chapitre.

L'intégration d'une nouvelle source se résume à développer un adaptateur pour cette dernière et le rendre accessible par le médiateur [CLU 98].

La diversité des sources doit être prises en compte dans le développement de l'adaptateur spécifique. En effet, certaines sources offrent des capacités limitées d'interrogation, par exemple savoir exécuter des sélections mais pas de jointure. Alors, le médiateur doit prendre compte les capacités des adaptateurs, de façon à éviter de soumettre aux adaptateurs des requêtes non traitables.

2.4.10. Génération de systèmes

A partir de l'architecture de référence définie ci-dessus, différentes options sont possibles. Un choix important est celui du modèle pivot et du langage pivot: XML est idéal, mais d'autre modèles ont été utilisés pour fédérer les données. Historiquement, il est possible de distinguer trois générations de systèmes de médiation.

2.4.10.1. La génération relationnelle

La première génération (dans les années 1975-1982) a été développée autour du modèle relationnel. Il s'agit des systèmes relationnels répartis centrés autour d'un SGBD (tels que Oracle et DB2 répartis) qui joue le rôle de médiateur. Ces systèmes répartis permettent un degré limité d'hétérogénéité par des passerelles qui adaptent les autres SGBD.

2.4.10.2. La génération relationnelle étendue

La seconde génération (dans les années 1980-1987) est plus ambitieuse en matière d'hétérogénéité. Il s'agit des systèmes fédérés toujours construits autour du relationnel, mais pas centrés sur un SGBD particulier. L'objectif est de permettre un accès intégré à des bases locales présentant une large diversité syntaxique et sémantique.

OLE-DB de Microsoft peut être perçu comme le produit phare de fédération de données de deuxième génération. Il s'agit de véritable SGBD en mémoire s'appuyant sur des sources de données munies d'un adaptateur de type ensemble d'enregistrements (Record Set). Le médiateur répond à une interface de type SQL étendue avec le fameux ADO (Active Data Object) de Microsoft. OLE-DB s'étend aujourd'hui vers la troisième génération et XML avec OLE-DB.NET et ADO.NET.

2.4.10.3. La génération XML

Il en va différemment avec la nouvelle génération basée sur XML. En effet, des produits sont déjà commercialisés par des éditeurs reconnus telles BEA avec Liquid Data, Nimble.com, Enosys Software ou e-XMLMedia. Microsoft fait évoluer rapidement OLE-DB et l'interface ADO dans leurs versions .NET vers un médiateur XML.

Tous ces produits, à l'exception d'OLE-DB.NET qui est plus hybride, on en commun une architecture basée sur un modèle pivot XML, et ils offrent un langage de requête pour XML, typiquement XQuery.

2.5. INTEGRATION D'APPLICATIONS

Au-delà des bases de données, il faut fédérer les applications. Trois technologies au moins adressent ce besoin d'intégrer des applications hétérogènes dans de nouveaux systèmes d'information coopératifs et ouverts au Web : les EAI (Enterprise Application Integrator), les portails d'applications (Web Application Portal) et les suites B2B (Business to Business).

2.5.1. Les intégrateurs d'applications d'entreprise (EAI)

Les EAI sont des outils d'échange d'informations semi-structurées entre applications, permettant d'automatiser les processus d'échange et de gestion des documents commerciaux (bons de commande et factures, par exemple) entre des applications situées dans la même entreprise ou dans des entreprises différentes. Les EAI sont composés essentiellement par :

Le dictionnaire de métadonnées

Le dictionnaire de métadonnées contient en général tous les schémas des données échangées via l'EAI. Les *schémas importés* décrivent les données issues des applications productrices alors que les *schémas exportés* décrivent les données à destination des applications consommatrices. Les données échangées peuvent être des données relationnelles, objets ou semi-structurées. De plus, le dictionnaire contient aussi les règles de mapping permettant de convertir les données depuis un format exporté vers le format cible.

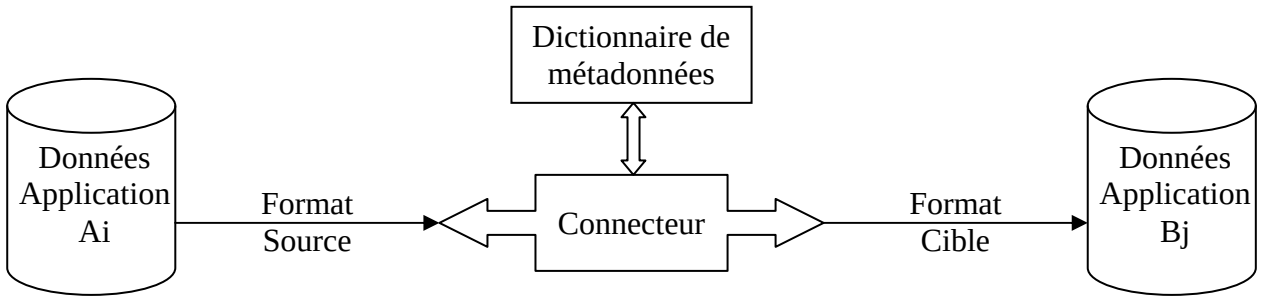


Figure 2. 6 - Formats de données lors d'un échange

Les connecteurs

Un connecteur (aussi appelés adaptateur) est un composant logiciel réalisant la connexion d'application avec l'EAI, et en utilisant le dictionnaire de métadonnées, il assure la traduction des données depuis le format source dans le format d'échange ou vice versa.

La figure 2.6 illustre l'échange de données entre deux applications Ai et Bj. Si n applications (A1, A2, …, An) communiquent avec m applications (B1, B2, …, Bm), l'approche EAI établit donc un point central pour l'échange de messages. En ce point central, les messages sont convertis dans un modèle de données global uniforme, de plus en plus souvent XML. Ceci permet un meilleur contrôle et réduit les conversions de format (comme nous avons vu plus haut dans ce chapitre), en plus, il assure une meilleur sécurité.

2.5.2. Intégration des applications et Portails d'entreprise

Le portail d'entreprise est un outil complexe d'intégration de données et d'applications, il permet un point d'accès web unifié à des applications hétérogènes via des pages web. Il fonctionne par émission et réception de messages XML de modules applicatifs hétérogènes à des vues homogènes implémentées par des connecteurs (figure 2.7). Ces vues se caractérisent par une représentation des applications existantes sous forme de services acceptant des messages bidirectionnels en XML.

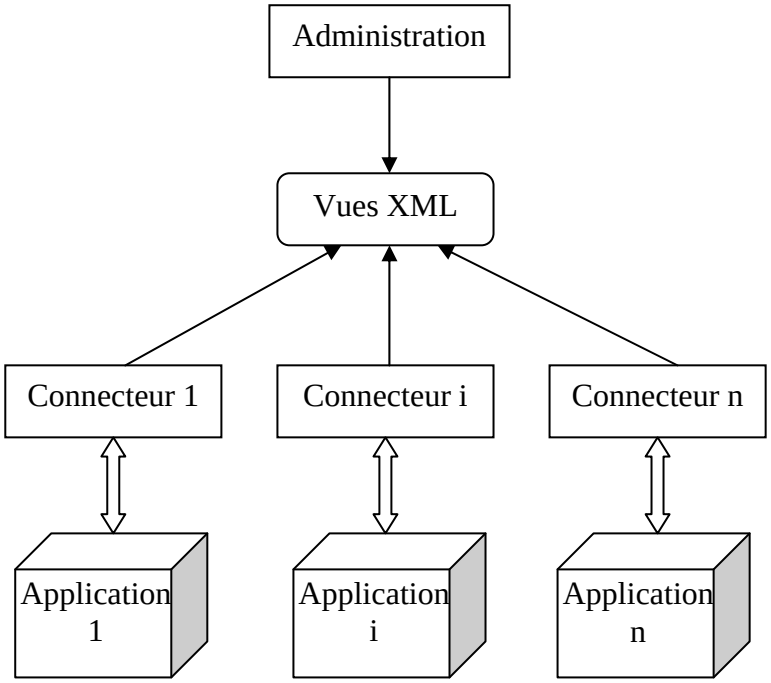


Figure 2. 7 - Architecture d'un portail d'application.

Connecteur d'application Web Service

Un connecteur d'application web service vise à faire apparaître une application existante comme un service web. Notamment, son interface doit être décrite en WSDL et doit être accessible via SOAP. Les applications sont en quelque sorte encapsulées par un

connecteur les transformant en service web. Le portail devient alors un intégrateur visuel de service web. Il est d'ailleurs lui-même un service web et peut être ainsi intégré au sein d'autres portails.

2.5.3. Intégration B2B et EDI XML

Une suite B2B reprennent l'EDI classique et la modélisation de processus d'affaire dans un cadre plus vaste basé sur XML, elle se déploie sur Internet pour le commerce électronique collaboratif. Elle s'appuie de plus en plus souvent sur les services web et vise à gérer des interactions complexes entre entreprises telles qu'une passation de commande, un appel de livraison ou une facturation.

2.5.3.1. Principe de l'EDI

L'EDI (Electronic Data Interchange) est une technologie d'échange vise à formaliser les échanges entre donneur d'ordre et fournisseur afin d'optimiser par exemple les approvisionnements. Elle permet d'améliorer la traçabilité des produits, etc., et plus généralement, d'assurer une réactivité plus forte des organisations. Ainsi, l'EDI permet de créer un environnement ouvert où tout partenaire peut facilement s'intégrer, en échangeant sous forme de messages codés et standardisés commandes, accusés de réception, demande d'état, factures, catalogues, demande de prix, positions de compte, etc.

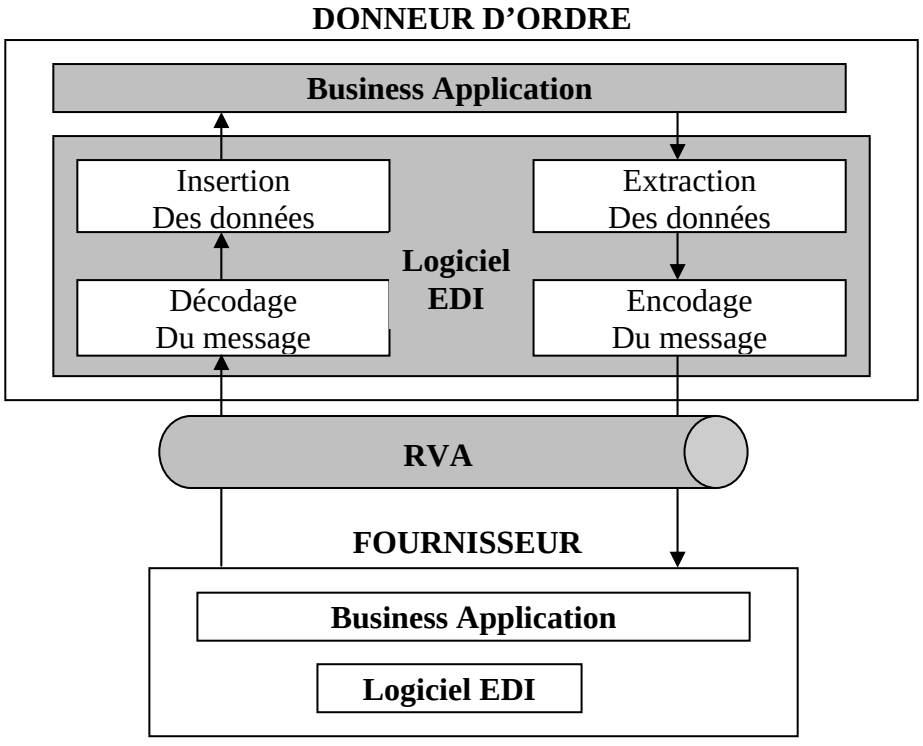


Figure 2. 8 - Aperçu de l'architecture de l'EDI classique.

Les opérations d'extraction et d'insertion de messages dans les systèmes d'information des partenaires des deux côtés sont automatisées (figure 2.8). Pour cela, un logiciel EDI réalise l'extraction des données depuis les bases du donneur d'ordre et la traduction en message EDI et vice versa. Un logiciel EDI est nécessaire pour le donneur

d'ordre et pour le fournisseur. Les messages sont échangés sur des réseaux à valeur ajoutée (RVA) type X400. Ceux-ci sont générés par de grands opérateurs ; ils garantissent la sécurité et la livraison. Les échanges sont sécurisés et ne peuvent en théorie être ni perdus, ni espionnés ou livrés deux fois, ceci grâce au RVA. Il est aussi possible de consulter des journaux des messages transmis.

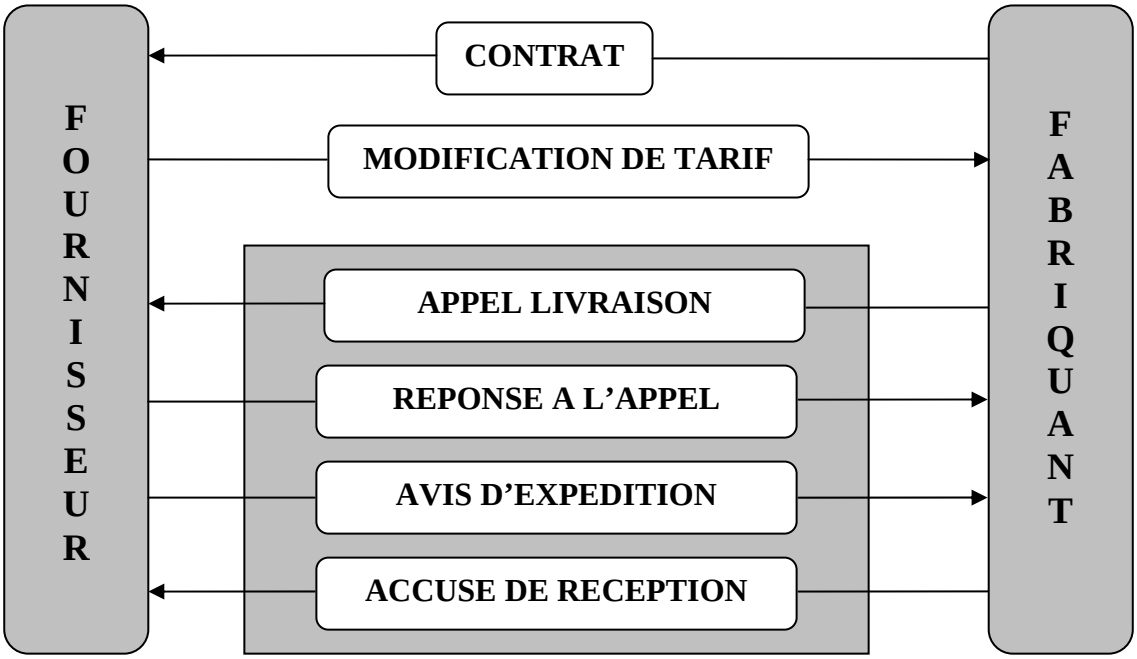


Figure 2. 9 - Exemple de transaction EDI.

Une transaction EDI permettra par exemple d'effectuer un appel de livraison sur catalogue depuis un fournisseur vers un fabriquant (figure 2.9). La transaction comprendra plusieurs messages allant dans les deux sens comme appel de livraison, réponse à l'appel, avis d'expédition, accusé de réception. Chaque message est composé de segments contenant des données de contrôle ou des données utilisateurs.

2.5.3.2. Intérêts et principes de l'EDI XML

- **Démocratiser l'EDI**
  L'objectif premier est de démocratiser l'EDI grâce à Internet, en s'affranchissant du réseau à valeur ajoutée (RVA) tout en assurant la sécurité et la fiabilité requises. En effet, le RVA est en général coûteux. L'utilisation d'Internet et de http pour transporter les messages EDI permettrait à la fois de réduire les coûts et de démocratiser l'accès. Ceci nécessite cependant d'assurer les services des RVA, à savoir authentification, confidentialité des messages, garantie de livraison et journalisation des messages échangés.

- **Traduire les messages EDI en XML**
  Le deuxième objectif est de profiter de la mouvance XML pour bénéficier de messages claires, lisibles et extensibles. Il s'agit de traduire les messages EDI en schéma XML. La traduction d'un diagramme de message EDI en schéma XML consiste à traduire l'arbre en schéma XML. En plus, on bénéficie de tous les outils XML pour le représenter et le traiter. Ceci illustre un autre avantage de passer l'EDI en XML.

2.6. INTEGRATION DE PLATES FORMES

Au-delà des bases de données et applications, il faut fédérer les plates formes. L'architecture Web Service vise à permettre aux applications de communiquer facilement via Internet ou intranet, quels que soient le système d'exploitation et le langage de programmation. Des modules communicants doivent pouvoir être implémentés sur toute plate-forme et échanger facilement de l'information en suivant des standards établis par le W3C. Les modules applicatifs vont pouvoir invoquer d'autres modules en ignorant leurs structures internes, uniquement via leurs interfaces exposées dans le langage de l'appelant (figure 2.10).

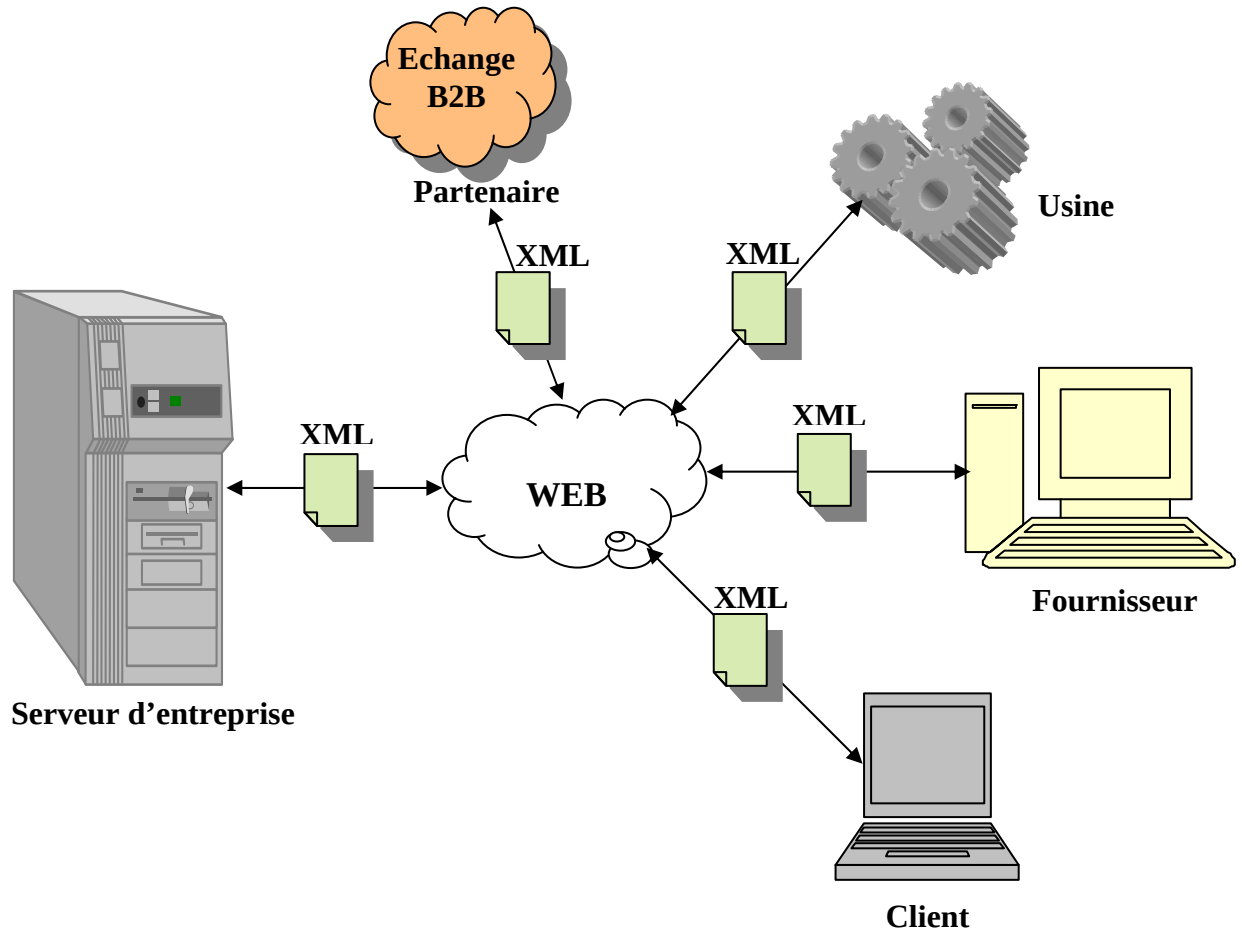


Figure 2. 10 - interface XML universel cachant les détails d'implémentation.

2.6.1. Qu'est-ce qu'un service web ?

Un service web [GAR 02] est un module applicatif exposé sur le web offrant une interface programmable accessible par XML. L'architecture service web présente quelques similarités avec l'architecture CORBA de l'OMG pour les objets distribués. Comme celle d'un objet CORBA, l'interface d'un service web doit être décrite dans un langage particulier appelé WSDL (Web Service Definition Language) afin d'être exposée sur le web. WSDL est un langage similaire à IDL de CORBA (Interface Definition Language), mais basé sur XML : c'est un langage de la famille XML de syntaxe un peu complexe. Les services web sont invoqués par le protocole SOAP.

2.6.2. Vue d'ensemble de SOAP

SOAP fournit à l'application cliente les outils nécessaires pour invoquer des services distants en lui donnant l'illusion qu'ils sont locaux. Grâce à WSDL, les applications utilisant SOAP sont capable de générer les souches de logiciels réalisant les échanges entre service Web. Pour le client, un compilateur WSDL doit être capable de générer le code d'appel du service, alors que pour le serveur il génère le code réalisant l'interface entre le protocole et le service. En claire, les deux souches encapsulent véritablement le service en masquant les détails techniques de bas niveau.

SOAP est un protocole indépendant de toutes plate-forme et de tout langage de programmation. Il réalise le codage des appels et retours de procédures en XML, ceci est plus général que les codages particuliers proposés par CORBA (IIOP/CDR), JAVA RMI, DCOM.

La figure 2.11 illustre avec plus de détails l'invocation d'un service Web par SOAP. Les messages SOAP générés par la souche lors de l'appel sont transférés vers un port de communication. En réponse, ils circulent de puit le service vers l'application. Les messages peuvent être échangés sur HTTP permettant la traversée des pare-feu et des contrôles associés. Il sont encapsulés dans un élément de haut niveau appelé enveloppe.

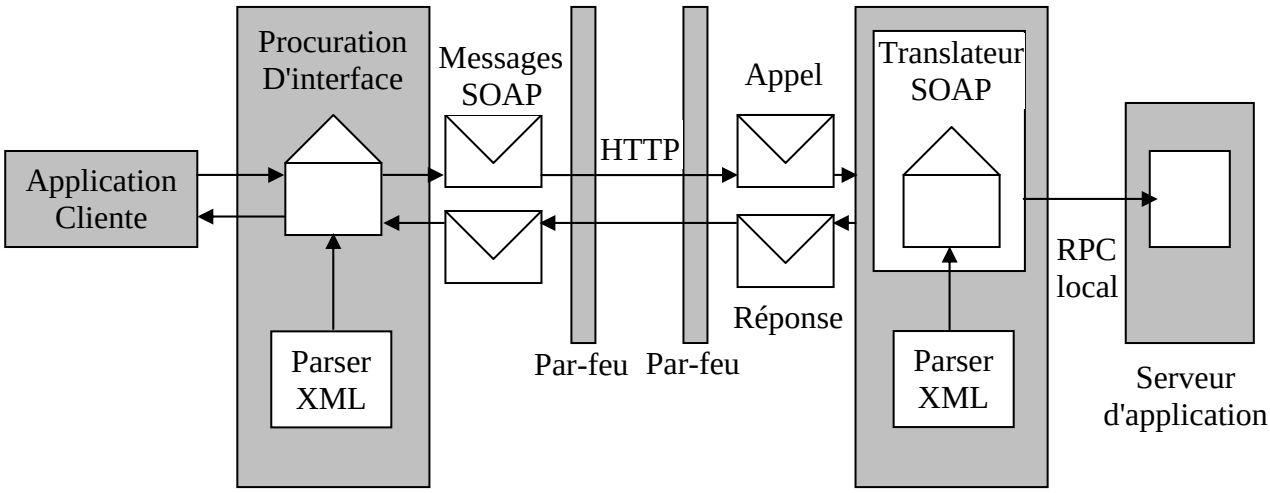


Figure 2. 11 - Schématique d'un échange SOAP

La racine du document XML échangé est <Envelope>. Tous les éléments définissant la structure des messages SOAP doivent être associés à l'espace de nom: "http://www.w3.org/2001/09/soap-envelope". L'élément enveloppe possède un ou deux fils dans l'ordre, l'élément optionnel <Header> et l'élément obligatoire <Body>. L'enveloppe est donc structuré comme suit:

```
<env:Envelope xmlns:env="http://www.w3.org/2001/09/soap-envelope">
      <env:Header>
        …
      </env:Header>
      <env:Body>
      …
      </env:Body>
</env:Envelope>
```

L'en-tête de message

L'en-tête de message <Header> est prévu pour étendre SOAP de manière modulaire et optionnelle. Il permet de passer à un partenaire des informations de contrôle par exemple pour l'authentification (nom et mot de passe), le contexte de transaction (identifiant et numéro de message), le paiement (carte de crédit), etc.

Le corps de message

L'élément <Body> contient l'information effective destinée au receveur. C'est lui qui contiendra les appels de procédures ou les rapports d'erreur. Il peut comporter de 0 à N élément fils appelés blocs, ceux-ci peuvent être qualifiés par un ou plusieurs espace de nom et par un style de codage (attribute encoding-style). Voici un exemple de corps permettant d'invoquer la méthode *QuelCours* du service *Bourse*.

```
<env:Body>
      <bourse:QuelCours xmlns:bourse='http://exemple.com/bourse">
            <CodeAction> 2017 </CodeAction>
      </bourse: QuelCours>
<env:Body>
```

2.6.3. Les règles de codage SOAP

Les règles de sérialisation des données en messages définies par SOAP sont spécifiées à l'URL "http//www.w3.org/2001/09/soap-encoding". Les messages SOAP utilisant ce type de codage doivent spécifier cette URL comme valeur de l'attribut *encodingStyle*. Des règles spécifiques à l'utilisateur peuvent être utilisées ou ajoutées.

Un élément de type simple codé avec les règles standard apparaît directement entre balises. Un élément complexe est codé soit comme une structure à balise, soit comme un

tableau dont les éléments sont accessibles par indices. Pour échanger des éléments complexes en dehors de SOAP, par exemple des objets binaires longs (BLOB), il est possible d'inclure dans un message HTTP un document SOAP et un contenu en format MIME (GIF, JPEG…). Il est possible encore de transférer des pièces jointes associées à un message SOAP.

2.6.4. Exemple de messages SOAP

Dans cet exemple, nous donnons les messages échangés sur HTTP pour obtenir une réservation dans un hôtel. Le premier message permet d'invoquer la fonction *Reserver* pour un hôtel particulier.

```
POST  /Hotel  HTTP/1.1
Host: http://www.hotel.com
Content-Type: text/xml; charset="utf-8"
Content-length: nnnn
SOAPAction: http://www.hotel.com/hotel#Resever

<env: Envelope xmlns:env="http://www.w3.org/2001/09/soap-envelope">
      <env:Body>
            <h:reserverCall xmlns:h="www.hotel.com/Hotel">
                  <IDHotel>212</IDHotel>
                <RDate>19-02-2002</RDate>
                  <Duree>7</Duree>
            </h:Reserver>
      </env:Body>
</env:Envelope>
```

Le second message ci-dessous est la réponse positive (le booléen *Reservation* est codé 1):

```
http / 1.1 200 OK
Content-TYPE: text/xml; charset="utf-8"
Content-Length: nnnn
<env:Envelope xmlns:env=http://www.w3.org/2001/09/soap-envelope>
<env:Body>
            <h:ReserverReturn xmlns:h=www.hotel.com/Hotel>
                  <RESERVATION>1</RESERVATION>
            </h:ReserverReturn>
      </env:Body>
</env:Envelope>
```

2.6.5. Gestion des transactions dans les services web

Les services web effectuent généralement des mises à jour de bases de données par le biais d'opérations applicatives, par exemple pour réserver un hôtel ou passer une

commande. Ils s'exécutent dans un contexte de systèmes distribués avec pannes sur Internet ou intranet. Il faut donc assurer le support de transactions fiables. Ceci nécessite l'intégration d'un protocole transactionnel à SOAP.

Une transaction est composée d'une suite de requêtes à la base qui doivent vérifier les propriétés d'atomicité, de cohérence, d'isolation et de durabilité, résumées par le vocable ACID. Nous synthétisons ces propriétés ci-dessous :

- **Atomicité :** Une transaction doit effectuer toutes ses mises à jour ou ne rien faire du tout. En cas d'échec, une transaction doit annuler toutes les modifications qu'elle a engagées.
- **Cohérence :** La transaction doit faire passer toute base de données d'un état cohérent à un autre. En cas d'échec, l'état cohérent initial doit être restauré.
- **Isolation :** Les résultas d'une transaction ne doivent être visibles aux autres transactions qu'une fois la transaction validée, ceci afin d'éviter les interférences avec les autres transactions.
- **Durabilité :** Dès qu'une transaction valide ses modifications, le système doit garantir que ces modifications seront conservées en cas de panne.

Dans le cadre d'une application invoquant plusieurs services web effectuant par exemple des mises à jour de données, le problème essentiel à résoudre est le risque d'incohérence lié au contrôle réparti : chaque site peut décider de valider ou d'annuler une opération. Il faut donc coordonner les validations. Le protocole de validation en deux phases a été proposé dès 1976 afin de coordonner l'exécution des commandes « COMMIT » par tous les sites participants à une transaction. Le contrôle du système réparti est centralisé sous la direction d'un site appelé *coordinateur*, les autres étant des *participants.*

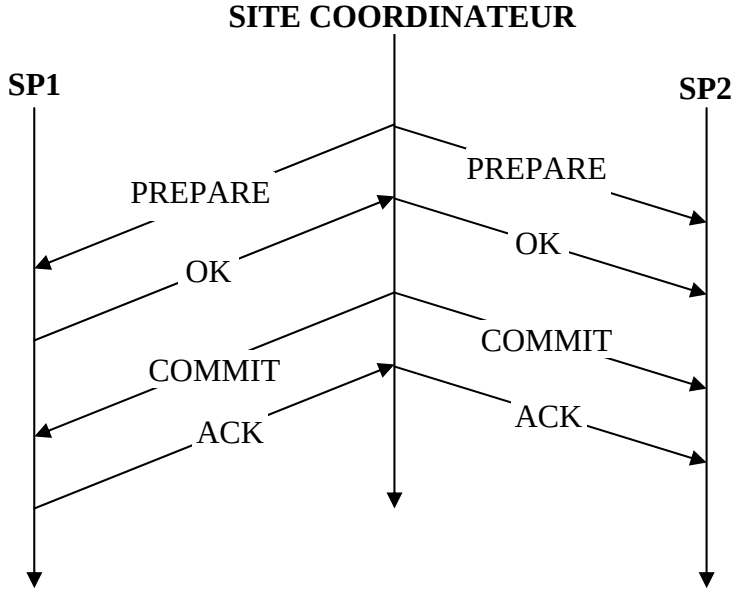


Figure 2. 12 - Validation en deux étapes avec XA

Ce protocole a été standardisé par l'ISO et le X/Open Group sous le nom XA [GAR 02]. XA offre un mécanisme standard pour coordonner les modifications à de multiples ressources (souvent des bases de données). XA est un protocole permettant de demander à chaque gestionnaire de ressources de voter avant d'accepter la validation. Le commit est effectué seulement si tous les participants votent « OK » suite au message de préparation (figure 2.12).

2.6.6. Sécuriser les services web

Un autre problème important pour les services web est d'assurer la sécurité des invocations et des données transportées via SOAP. Au-delà du nom d'utilisateur et d'un mot de passe transportés dans l'en-tête des messages SOAP pour permettre l'authentification, il faut être capable de chiffrer finement les données et de signer les messages.

2.6.6.1. Le chiffrement de messages XML

Le chiffrement XML (XML Encryption) est un standard permettant d'échanger des éléments cryptés ainsi que les clés publiques de cryptage et les références aux algorithmes nécessaires au déchiffrement. Pour illustrer cela, voici un exemple d'information décrivant un paiement de Jean Dupont par une carte de crédit :

```
< ?xml version= "1.0" ?>
<PaymentInfo xmlns= "http://example.fr/payment">
      <Name>Jean Dupont</Name>
      <CreditCard Limit="5,000" Currency="EURO">
            <Number>4019 2445 0277 5567</Number>
            <Issuer>Bank of the Internet</Issure>
            <Expiration>04/02</Expiration>
      </CreditCard>
</PaymentInfo>
```

Si l'on choisit de crypter toutes les informations concernant la carte de crédit, on obtiendra:

```
< ?xml version="1.0" ?>
<PaymentInfo xmlns= "http://example.fr/payment">
      <Name>Jean Dupont</Name>
      <EncryptedData Type="http://www.w3.org/2001/04/xmlenc#Element"
                        Xmlns="http://www.w3.org/2001/04/xmlenc#''>
            <CipherData>
                  <CipherValue>A23B45C56</ CipherValue>
            </CipherData>
      </EncryptedData>
</PaymentInfo>
```

2.6.6.2. La signature des messages XML

Une signature XML (XML Signature) est une partie de message XML standardisé permettant d'identifier de manière sûre l'émetteur d'un élément XML, consistant à échanger une combinaison de l'élément et d'un secret (constituant la signature), la description des informations signées et des clés nécessaires à la validation. D'un point de vue fonctionnel, la signature d'un élément permet la non répudiation : l'émetteur ne peut pas contester l'envoi du message.

L'information effectivement signée est décrite dans un élément <SignedInfo>. La signature suit cet élément balisé par <SignatureValue>, puis suivent les informations sur les clés ayant servi à son calcul dans un élément <KeyInfo>. A ce point, une signature est donc un document XML composé comme suit :

```
<Signature Id= "MaSignature" xmlns="http://www.w3.org/2000/09/xmldsig#">
      <SignedInfo>
            … < !-- Description des objets signés --> …
            … < !-- Description de la signature --> …
      </SignedInfo>
      <SignatureValue>MC0CFFrVLTRlK= … </<SignatureValue>
       <KeyInfo>
            … < !-- Definition des clés -->…
      </KeyInfo>
<Signature>
```

3.1. INTRODUCTION

L'Internet a permis une augmentation considérable du nombre et du type de sources de données. Par conséquent, un médiateur devient essentiel afin de fournir un accès unifié à des sources de données hétérogènes sur le Web. Par exemple, un utilisateur cherchant à réserver un voyage a besoin d'un système d'intégration qui va chercher dans diverses sources (site d'agence de voyage, site de chaîne hôtelière, horaire d'avions, logiciel d'agenda personnel) et produit un résultat intégré. L'intégration des sources de l'Internet pose de nouveaux problèmes:

- *Les données semi structurées:* les données ne possèdent pas de structures fixes. Les traitements classiques d'évaluation et d'optimisation ne s'applique plus;
- *L'hétérogénéité des données:* les données peuvent être aussi bien semi-structurées que relationnelles, objets ou textuelles. Comment gérer la diversité des schémas donnés par les différentes sources? Comment intégrer les résultats de différentes sources?
- *La communication avec des sources hétérogènes:* les sources sont accessibles de diverses manières. Certaines font appel à des interfaces de requêtes évoluées (ODBC, JDBC, IIOP) avec des langages de requêtes adaptés (SQL, OQL, XQuery) d'autre au contraire ne disposent que de moteurs de recherche ou d'interface de programmation très spécifique.

À l'instar de médiateurs existants (voir l'annex), nous adaptons l'architecture à base de médiateurs et adaptateurs (DARPA I3 [WIE 92]).

Un médiateur est un composant qui traite de la *distribution* des données. Il interagit avec les différentes sources par l'intermédiaire d'un langage commun. L'intégration des différents résultats se fait par un traitement local au niveau du médiateur.

Pour permettre au médiateur d'interagir facilement avec les différents types de sources, l'interaction est faite par un module logiciel nommé adaptateur. L'adaptateur traite de l'hétérogénéité des sources. Il traduit le langage commun utilisé par le médiateur en langage spécifique à la source à laquelle il est connecté. De la même façon, les résultats rendus par la source sont traduits dans un modèle de données commun utilisé par le médiateur

Outre le passage des requêtes et des résultats, l'interaction entre le médiateur et l'adaptateur peut aussi servir à communiquer des informations relatives aux sources de données. Ces informations peuvent être:

- les déscriptions des métadonnées: c'est-à-dire les métadonnées ou les schémas gérés par la source;
- des descriptions de capacités des sources: c'est-à-dire les types de requête que la source est capable de traiter.

L'évaluation d'une requête au sein du médiateur est un traitement complexe. La requête doit être analysée. Les sources intervenant dans le traitement de la requête doivent être identifiées. La requête doit être décomposée en différentes sous-requêtes exécutables chacune par les sources de données. Une phase d'optimisation est appliquée pour améliorer le temps de traitement. Enfin, après une phase de recomposition des résultats, le résultat finale peut être envoyé à l'utilisateur.

3.2. ARCHITECTURE DU SYSTÈME DE MÉDIATION

La figure 3.1 décrit l'architecture médiateur/adaptateur de notre prototype.

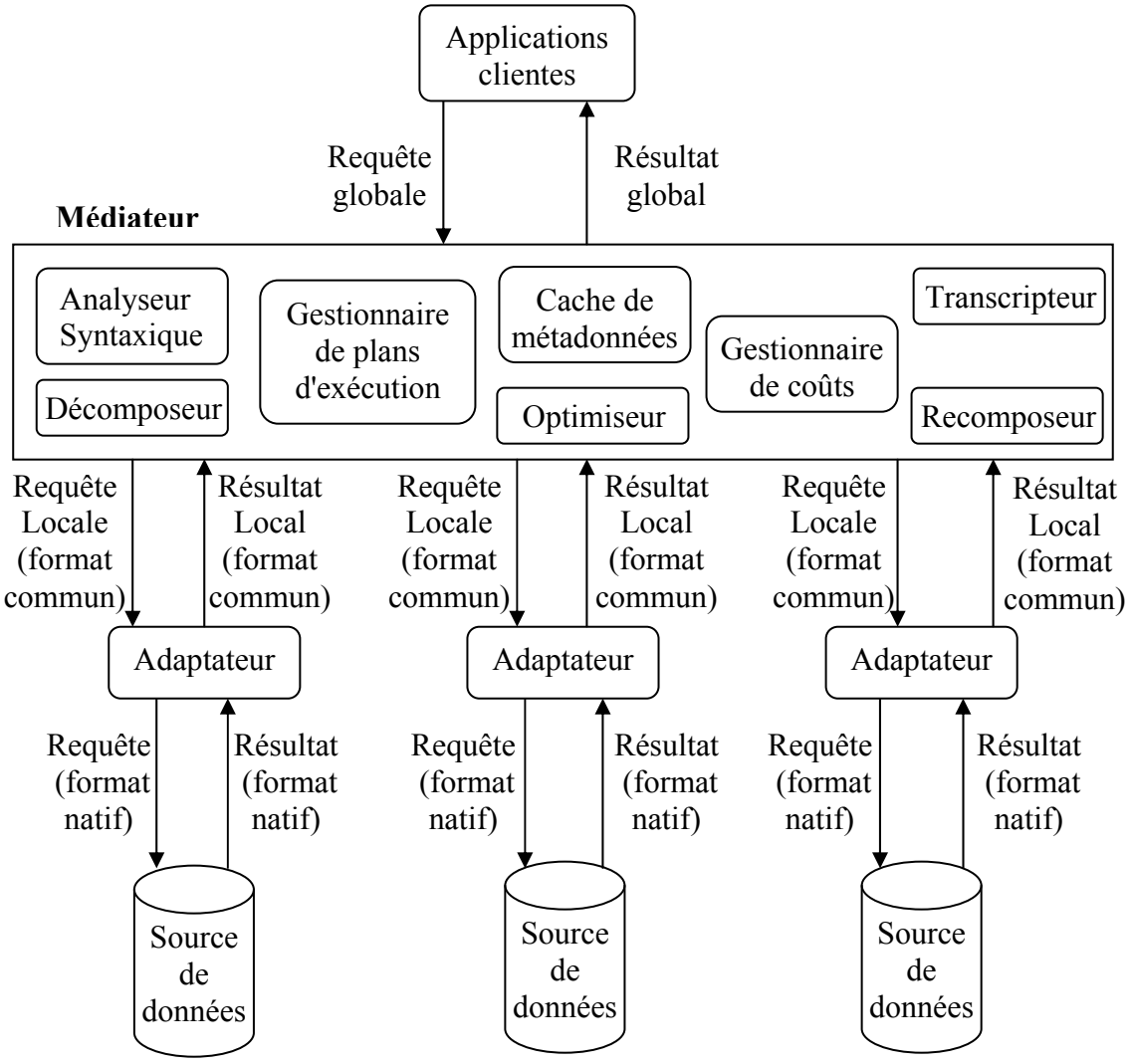


Figure 3. 1 - Architecture générale du médiateur

Au niveau le plus bas se situent les sources de données qui stockent et manipulent les données. Au dessus de chaque source est connecté un adaptateur. Le rôle d'un adaptateur est de masquer les détails de l'interface de la source de données. Le cœur de l'architecture est le médiateur. Ce médiateur est décomposé en plusieurs modules:

- l'analyseur/décomposeur: il analyse la structure de la requête afin de déterminer comment décomposer la requête en une structure interne susceptible d'être manipuler facilement par les différents composants du médiateur. Il vérifie aussi si la requête est syntaxiquement valide par rapport aux types de données interrogées;
- le cache de métadonnées: 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 de plans d'exécution: il permet de générer l'ensemble des plans d'exécution possibles pour satisfaire une requête donnée;
- l'optimiseur: il détermine en fonction des composants ci-dessus, quel est le plan d'exécution optimal pour évaluer une requête;
- le composeur: il restructure et recompose les résultats données par les différents adaptateurs;
- le transcripteur: il transforme la structure interne résultat en un format lisible par l'utilisateur;
- un gestionnaire des connexions et des échanges responsables des connexions avec les adaptateurs, de la transmission des sous-requêtes à ces derniers et de la réception des résultats.

3.2.1. Modèle de données

La plupart des études sur les systèmes de gestion de données distribuées hétérogènes utilisent le modèle relationnel ou le modèle objet comme modèle de données d'intégration. Notre approche est d'utiliser le standard XML [BRA 98] comme modèle de données d'intégration.

Les avantages de l'utilisation de XML comme modèle d'intégration tiennent à la richesse de ce langage: abondance des descriptions et typage des données, clarté, extensibilité. Un autre critère participant au choix de XML comme modèle d'intégration est que son caractère général rend aisée la traduction des modèles de données existants. Pour chaque objet du modèle réseau, héararchique, 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 hétérogènes. Ainsi XML-Shema permet de fournir un modèle de métadonnées uniformes entre les adaptateurs et le médiateur, le langage XQuery offre l'interrogation efficace de requête sur XML, et les feuilles de style XSL permettent de présenter les données à l'utilisateur.

3.2.2. Traitement des sources

Les différentes sources sont accessibles à travers des adaptateurs qui communiquent par des connecteurs des services web possédants des interfaces décrites en WSDL, et accessibles via SOAP. Il existe des outils pour générer à partir des fonctions publiques d’une application un descripteur WSDL. En utilisant ce descripteur, il est possible de générer ou de développer un adaptateur capable de recevoir les messages SOAP d’invocation, d’appeler les fonctions, de récupérer les réponses et de renvoyer les réponses en SOAP.

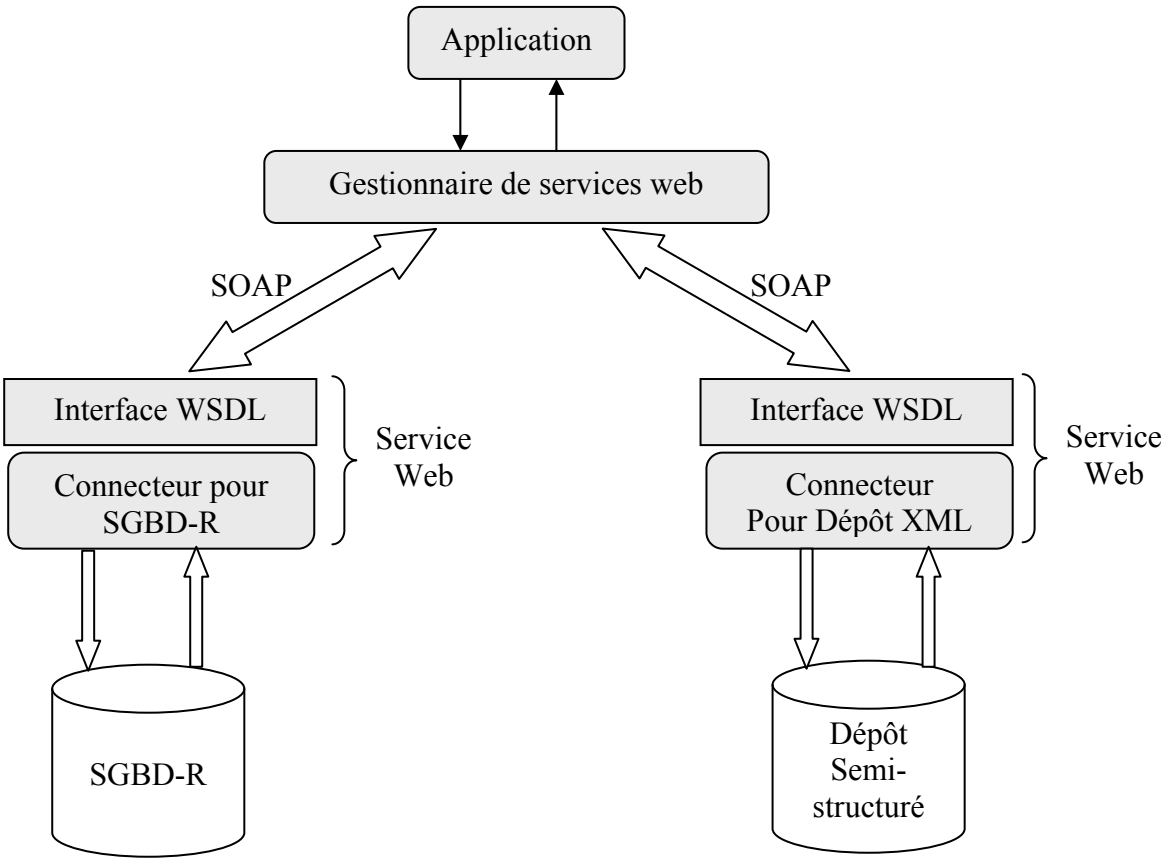


Figure 3. 2 - Accès par services web

La figure 3.2 illustre une application accédant à des sources de données via l'interface WSDL des services web. Cette interface permet à l'application d'adresser indifféremment à chacune des sources comme suit:

- ouverture de connexion suivant les protocoles associés. Les authentifications nécessaires sont éventuellement effectuées;
- demande d'informations sur les sources: lors de leur initialisation, les sources peuvent communiquer différentes informations au médiateur:
  - informations sur les métadonnées: la structure et les types des données gérées par la source (schéma, tables);
  - informations sur les capacités des sources: chaque source n'est capable de traiter que certains type de requête. Ces informations de capacité doivent être communiquées au médiateur;
  - informations sur les formules et statistiques de coût: qui permettent au médiateur d'estimer les temps d'évaluation des requêtes.
- exécution de requête XQuery;
- récupération du résultat en SAX ou en DOM;
- fermeture de connexion.

3.2.3. Définition des sources de données

Les différentes sources de données doivent être accédées via un adaptateur dédié. L'ensemble des adaptateurs associés à un médiateur est spécifié lors de l'initialisation à l'aide d'un fichier de configuration au format XML.

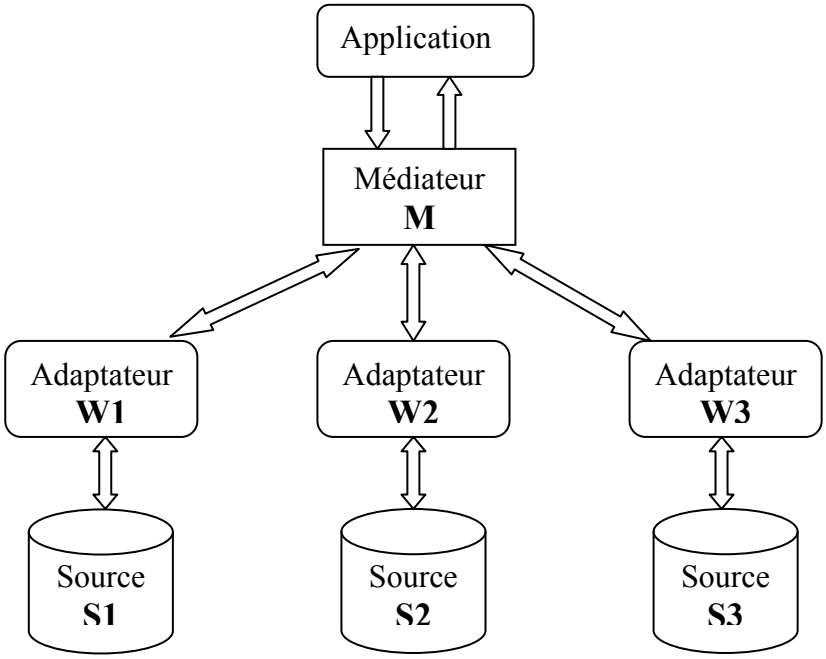


Figure 3. 3 - Interconnexion de médiateur / adaptateurs

Le fichier de configuration du médiateur M de la figure 3.3 est représenté par le document XML suivant:

```
<mediateur nom="M">
  <adaptateurs>

  <adaptateur nom="w1" type=”Relationnel”>
     <connexion>
        <chemin>http://.../w1.wsdl</chemin>
    </connexion>
  </ adaptateur >

 < adaptateur nom="w2" type=”Relationnel”>
    <connexion>
        <chemin>http://.../w2.wsdl</chemin>
        <login>USER1</login>
       <passwd>PASSW1</passwd>
    </connexion>
  < / adaptateur >
```

```
< adaptateur nom="w3" type=”XML”>
    <connexion>
        <chemin>http://.../w3.wsdl</chemin>
    </connexion>
  < /adaptateur>

< / adaptateurs>
```

Chaque méthode d'accès à un adaptateur est représentée par l'élément *adaptateur*. Elle est identifiée par un nom (valeur donnée par l'attribut **nom** de l'élément *adaptateur)*, et par un type qui présente le type de la source de données (relationnel, semi-structurées, …). Elle est accessible par l'intermédiaire d'une interface WSDL spécifique à la source accédée. L'adaptateur est localisé sur le réseau par une URL (Uniform Ressource Locator) indiqué par l'élément *chemin.*

3.2.4. Exportation d'informations

Lors de l'initialisation l'adaptateur communique des informations au médiateur permettant à ce dernier de définir les décisions à prendre pour l'évaluation des requêtes. Ces informations sont: les schémas gérés par les différentes sources, les capacités des sources, et enfin les statistiques et les formules de coût d'évaluation des requêtes sur ces adaptateurs.

3.2.4.1. Exportation de métadonnées

Lorsqu'un nouvel adaptateur est enregistré auprès du médiateur, ce dernier lui demande le schéma de la source via une méthode de l’interface WSDL **getMetData**().

L'adaptateur envoie alors le schéma demandé sous forme d'un document XML encapsulant un ou plusieurs documents XML-Schema. Ce document XML est appelé description de métadonnées.

À titre d'exemple, la description de métadonnées de l'adaptateur w3 de la figure 3.3 comporte deux schémas définis comme suit:

- L'adaptateur w3 comporte le schéma d'une collection **livre**.

```
<xs:schema xmlns:xs="hhtp://www.w3.org/2001/XMLSchema"
    targetNamespace="http://www.bibliographie.org"
    xmlns="http://www.bibliographie.org">
  <xs:element name="livre">
    <xs:complexType>
      <xs:sequence>
        <xs:element name="titre"  type="xs:string"/>
        <xs:element name="numero"  type="xs:decimal"/>
        <xs:element name="date"  type="xs:dateTime"/>
        <xs:element name="auteur">
          <xs:sequence>
```

```
                <xs:complexType>
                 <xs:element  name="nom"  type="xs:string"/>
                 <xs:element  name="prenom"  type="xs:string"/>
                <xs:complexType/>
               <xs:sequence/>
       </xs:element>
     </xs:sequence>
  </xs:element>
</xs:schema>
```

- L'adaptateur w3 comporte également le schéma d'une collection **personnes**.

```
<xs:schema xmlns=http://www.w3.org/2001/XMLSchema
     targetNamespace="http://www. personnel.org"
      xmlns="http://www.personnel.org">
   <xs:element name="personne">
     <xs:complexType>
       <xs:sequence>
          <xs:element name="nom"  type="xs:string"/>
          <xs:element name="prenom"  type="xs: string "/>
          <xs:element name="date_nais"  type="xs:dateTime"/>
          <xs:element name="adresse">
           <xs:sequence>
                <xs:complexType>
                 <xs:element  name="rue"  type="xs:string"/>
                 <xs:element  name="ville"  type="xs:string"/>
                <xs:complexType/>
               <xs:sequence/>
         </xs:element>
      </xs:sequence>
  </xs:element>
</xs:schema>
```

Les deux schémas décrits ci-dessous sont encapsulés dans un seul document XML dont son en-tête s'écrit à la façon suivante:

```
<?xml version='1.0'?>
<xm:metadata
        xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
        xmlns:xm="http://LacalHost/Metadata"
        source="SourceW3">
```

Le médiateur ayant interrogé tous les adaptateurs auxquels il a accès sur leurs schémas respectifs, la figure 3.4 représente la description de métadonnées des adaptateurs w1, w2 et w3.

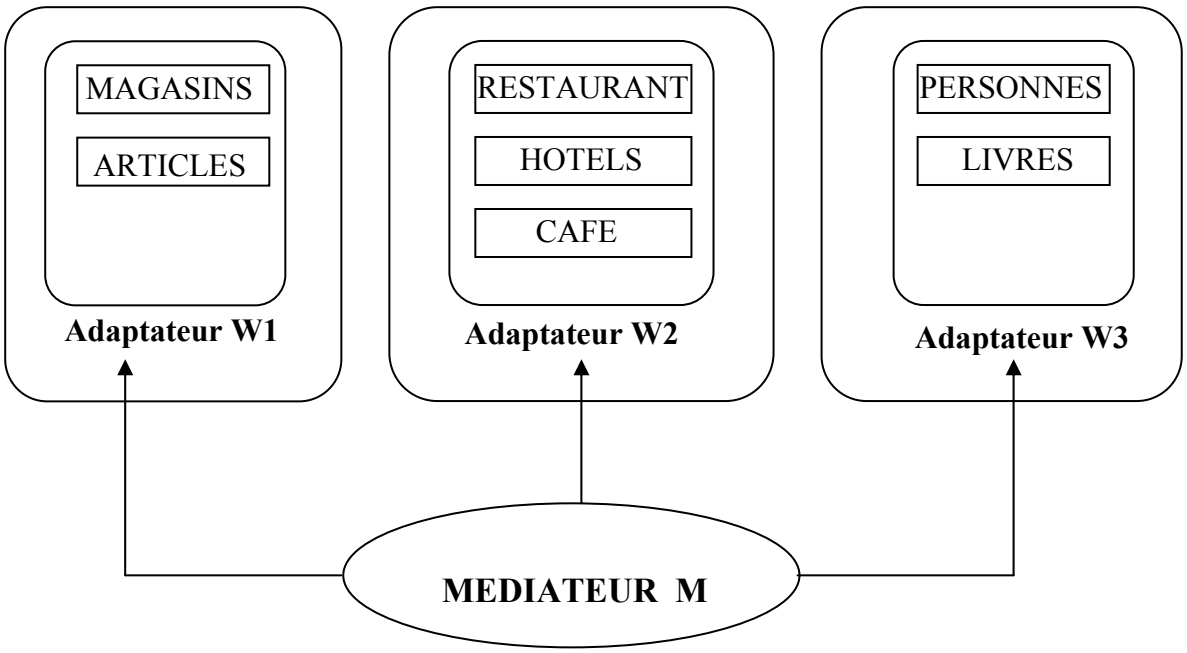


Figure 3. 4 - description de métadonnées des sources gérées par W1, W2 et W3.

3.2.4.2. Exportation de capacité des sources

Un adaptateur est muni de capacités permettant de décrire ce dont il est capable en terme d'opérateur de base: jointure entre collection, tri de résultats, calcule d'agrégats, etc. Un médiateur ne doit déléguer qu'en fonction des capacités d'un adaptateur. Par exemple, un SGBD relationnel fournit une interface avancée, permettant des requêtes complexes d'interrogation. Une page Web au contraire, est une source avec une interface très pauvre ne permettant que des requêtes très limitées.

Lorsqu'un nouvel adaptateur est enregistré auprès du médiateur, ce dernier lui demande les informations de capacité de la source via une méthode de l'interface WSDL du service web **getCapacity**().

Liste de règles de capacités: une liste de règles de capacités [NGO 03] d'un adaptateur servent à la vérification de la capacité d'un adaptateur à répondre à une requête. L'expression est alors comparée successivement à chaque règle de la liste. Lorsqu'une correspondance est trouvé, la réponse correspondant à la règle trouvé est alors envoyée. Cette réponse peut être soit *allow*, ce qui veut dire que la requête peut être pris en charge par l'adaptateur, soit *deny* dans le cas contraire.

Une règle de capacité se définit par au plus 8 champs appelés proprités:

1. *Le numéro de règle* (p1): les règles sont ordonnées suivant leur numéro de règle, et lues dans cet ordre lors de la vérification des règles.
2. *La permission* (p2)*:* positionnée soit à *allow* soit à *deny* indique si la requête donnée peut être prise en charge par l'adaptateur ou non.
3. *L'opérateur algébrique* (p3): l'opérateur algébrique sur laquelle s'applique la règle (*scan*, *project*, *join*, *select*).

4. Le nom de collection (p4): le nom de la collection sur laquelle s'applique l'opérateur algébrique (valable pour *scan*, *project*, *select*, *join*).
5. *Le chemin* (p5): le chemin (ou l'attribut dans le cas du relationnel) sur lequel s'applique l'opérateur.
6. *L'opérateur de comparaison* (p6): <, >, =, utilisé par l'opérateur (valable pour *join* et *select*).
7. *Le nom de collection* (p7): la collection avec laquelle la jointure est possible (valable pour *joint*).
8. *Le chemin* (p8): le chemin (ou l'attribut) de collection avec lequel la jointure est possible (valable pour *join*).

Dans une règle, seuls les mot-clef **allow** et **deny** sont obligatoires, les autres sont facultatifs (mais l'ordre et le type des arguments si ceux-ci sont existants, doivent respecter la syntaxe décrite précédemment). Ci-dessous, un exemple d'une liste de règles de capacités:

P1	P2	P3	P4	P5	P6	P7	P8
10	allow	Scan					
20	allow	Select	personne				
30	allow	Select	Voiture		=		
100	allow	Select	Voiture	Age	<		
200	allow	Project	Personne	Nom			
260	allow	Project	Voiture				
261	allow	join	Personne	Id	-	Voiture	Id_conducteur

Cette liste de règle de capacité se décrit de la façon suivante:
Règle 10: autoriser la lecture séquentielle pour toutes les collections;
Règle 20: autoriser toute sélection sur la collection personne;
Règle 30: autoriser la restriction sur la collection voiture si elle fait intervenir l'opération d'égalité;
Règle 100: autoriser la restriction sur l'attribut Age de la collection voiture si elle fait intervenir l'opération d'infériorité;
Règle 200: autoriser la projection sur l'attribut nom de personne;
Règle 260: autoriser la projection sur n'importe quel attribut de voiture;
Règle 261: autoriser l'équi-jointure entre l'attribut id de personne et l'attribut id_conducteur de voiture;

Les règles de capacité sont exportées par les adaptateurs sous forme de document XML. Ainsi, l'exemple de liste de règles décrites ci-dessus traduit par le document suivant:

```
<ruleset>
        <rule num="10">
                <permission> allow </permission>
                <relationalop> scan </relationalop>
        </rule>
        <rule num="20">
                <permission> allow </permission>
                <relationalop> select </relationalop>
```

```
            <collection1> personne </collection1>
        </rule>
        <rule num="30">
            <permission> allow </permission>
            <relationalop> select </relationalop>
            <collection1> voiture </collection1>
            <operator> equal </operator>
        </rule>
        <rule num="100">
            <permission> allow </permission>
            <relationalop> select </relationalop>
            <collection1> voiture </collection1>
            <attribute1> age </attribute1>
            <operator> less </operator>
        </rule>
        <rule num="200">
            <permission> allow </permission>
            <relationalop> project </relationalop>
            <collection1> personne </collection1>
            <attribute1> nom </attribute1>
        </rule>
        <rule num ="260">
            <permission> allow </permission>
           <relationalop> project </relationalop>
            <collection1> voiture </collection1>
        </rule>
         <rule num="261">
            <permission> allow </permission>
            <relationalop> project </relationalop>
            <collection1> personne </collection1>
            <attribute1> id </attribute1>
            <operator> equal </ operator>
            <collection2> voiture </collection2>
            <attribute2> id_conducteur </attribute2>
        </rule>
</ruleset>
```

Avant de passer une sous-requête à un adaptateur, on vérifie si celle-ci peut être prise en charge par l'adaptateur. Pour cela on applique l'algorithme suivant:

```
Pour chaque sous-requête R à appliquer sur un adaptateur A
        Appliquer la liste des règles de capacité de l'adaptateur A
  Si l'opérateur est accepté alors
        La sous-requête peut être envoyée à l'adaptateur
  Sinon
        Il devra être pris en charge par le médiateur
  Fin si
Fin pour
```

3.2.4.3. Exportation de statistiques et formules de coût des sources

Lorsqu'un nouvel adaptateur est enregistré auprès du médiateur, ce dernier lui demande des informations de coût de la source via une méthode de l'interface WSDL **getCost()**. Chaque adaptateur peut avoir des formules de coût et des statistiques qui lui sont spécifiques. Dans ce cas, il faut pouvoir intégrer ces formules de coût dans le coût global du médiateur. Afin que les adaptateurs puissent communiquer leurs informations au médiateur. Nous définissons un langage d'expressions mathématiques **MathML** [AUS 02]. Ce langage est une spécification du W3C qui permet de coder en XML à la fois la représentation (notation) et la structure (contenu) mathématique. La version 1.01 de ce langage date de juillet 1999 et une première version 2.0 a vu le jour en février 2001. En octobre 2003, la deuxième mouture de MathML Version 2.0 a été rendue publique comme la version finale du groupe de travail du W3C [WIK 06].

Une expression mathématique est composée d'opérateur, d'identifiants et de valeurs:

- L'élément ***cn*** permet de représenter des nombres entiers, rationnels, réels ou complexes;
- L'élément ***ci*** est utilisé pour représenter tous les identifiants. Par exemple les variables des fonctions;
- L'élément ***apply*** permet de grouper les opérateurs avec arguments;
- L'élément ***declare*** permet de définir une fonction ou une variable. Pour cela, on introduit l'identifiant de la nouvelle fonction ou variable par l'élément *ci*, et si c'est une fonction paramétrée les variables avec les éléments ***lambda*** et ***bvar***.

La **figure 3.5** montre comment exprimer l'expression **(a)** sous forme MathML **(c)**. **(b)** montre la décomposition intermédiaire de l'expression **(a)** sous forme d'arbre algébrique.

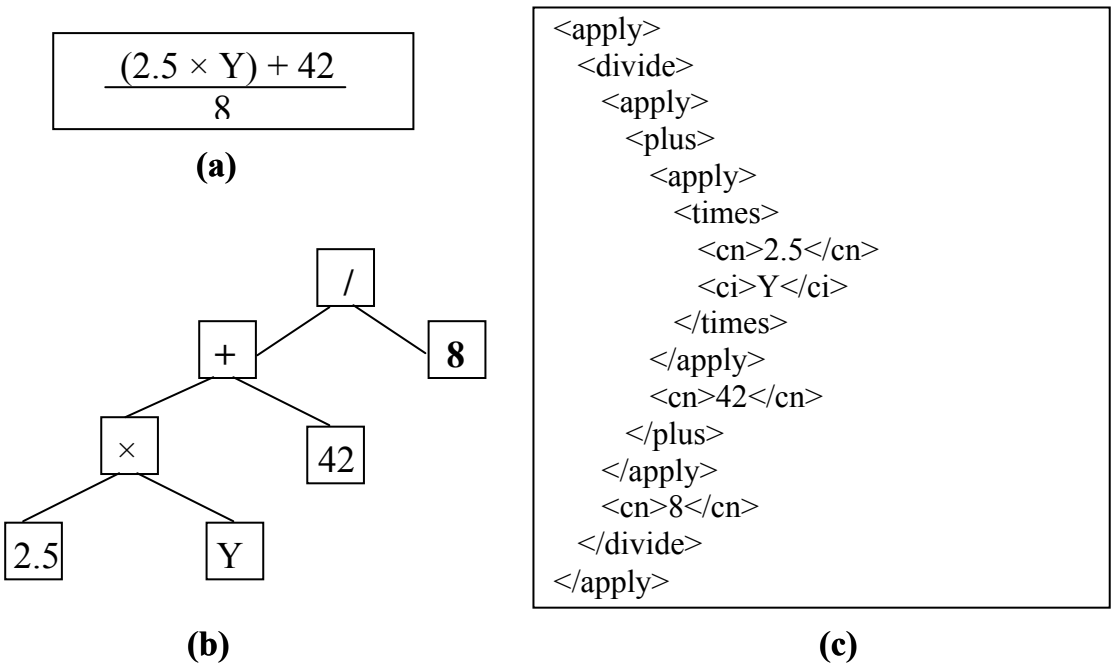


Figure 3. 5 - Expression mathématique en MathML

En utilisant MathML, on donne la possibilité de définir une fonction, d'affecter des variables et de réaliser des appels de fonctions et de variables.

Par exemple, on affecte au variable VAR1 la valeur 77, soit en pseudo-code: VAR1:=77, et en MathML:

```
<declare type="real">
        <ci> VAR1 </ci>
        <cn> 77 </cn>
</declare>
```

On définit en suite la fonction **ma_fonction** (x, y) = (x + VAR1 + y) -8.

```
<declare type ="fn">
        <ci> ma_fonction </ci>
        <lambda>
                <bvar>
                        <ci> x </ci>
                        <ci> y </ci>
                </bvar>
                <apply>
                        <minus>
                                <apply>
                                        <ci> x </ci>
                                        <ci> VAR1 </ci>
                                        <ci> y </ci>
                                </apply>
                                <ci> 8 </ci>
                        </minus>
                </apply>
        </lamda>
</declare>
```

Certains noms de variables et de fonctions ont néanmoins une signification particulière pour l'évaluation de coût, elles peuvent représenter des statistiques de données ou des formules de coût d'opérateurs d'algèbre.

On distingue les statistiques systèmes, les statistiques de collections et les formules de coût. L'exportation sous la forme d'un document XML se présentant sous la forme suivante:

```
<costmodel>
        <statistique>
                <system>
                        expressions mathématiques
                </system>
                <collection>
                        expressions mathématiques
                </collection>
```

```
            <user-defined>
                expressions mathématiques
            </user-defined>
        </statistique>
        <formulas>
            <generic>
                expressions mathématiques
            </generic>
            <operators>
                expressions mathématiques
            </operator>
        </formulas>
</costmodel>
```

Dans le document ci-dessus, on définit dans la clause <statistique>: les statistiques du système d'exploitation (CPU, E/S), les statistiques de collection et les variables définit par l'utilisateur. Et dans la clause <formulas>: les fonctions intermédiaires définies par l'utilisateur, les formules de coût génériques et en fin les formules de coût associées aux opérateurs algébriques (scan, project, etc.).

3.2.5. Exécution de requêtes

L'exécution de requête sur les sources se fait par l'intermédiaire de la méthode de l'interface WSDL **executeQuery (requete).** Les requêtes sont exprimées dans le langage XQuery.

Exemple: Renvoyer le nom de l'auteur et le titre des livres dont la date est postérieure à 1990 et dont le titre contient le mot "SOAP".

```
For $l in Collection("*:*") /livre
Where
        $l/date > 1990
    and
        contains($l/titre, "SOAP")
return
        <livre>
            <auteur> $l/auteur/nom </auteur>
            <titre> $l/titre </titre>
        </livre>
```

La réponse à la requête se présente sous forme d'un ensemble de documents XML (un par résultat trouvé), Le résultat de la requête ci-dessus sera:

```
<results>
        <livre>
            <auteur>Gardarin</auteur>
            <titre>Communiquer avec SOAP</titre>
        </livre>
        <livre>
```

```
            <auteur>Snell</auteur>
            <titre>Programming Web Service with SOAP</titre>
        </livre>
</results>
```

3.3. TRAITEMENT D'UNE REQUETE XQUERY

Il y a plusieurs façons de traiter une requête, chacune de ces façons étant appelée un *plan d'exécution*. L'ensemble des plans d'exécution pouvant être très grande, il est impossible d'explorer toutes les possibilités afin de déterminer le plan optimal. Il s'agit donc de proposer des heuristiques afin de restreindre cet ensemble en un ensemble plus petit appelé *espace de recherche*.

Le rôle de l'optimiseur est de déterminer l'espace de recherche, puis d'examiner chacune des possibilités d'exécution afin de choisir le plan optimal dans cet espace.

Le coût d'un modèle d'exécution peut s'exprimer en fonction de:

- temps d'exécution (temps observé entre le lancement de la requête jusqu'à l'obtention des résultats);
- travail (consommation de ressources, communication, place mémoire);

La manipulation des sources hétérogènes implique des traitements différents selon les sources. De ce fait, les algorithmes classiques des optimisations utilisés dans les bases des données traditionnelles ne s'applique plus à cause de l'absence de connaissances des propriétés des données manipulées (index, distribution, schémas, cardinalités).

La construction simple d'un plan d'exécution se fait suivant les étapes suivantes:

1. Normalisation et canonisation de la requête: il s'agit de transformer la requête suivant certaines règles d'équivalence afin d'obtenir une forme générique plus apte à être traiter dans la suite de nos opérations.
2. Atomisation des requêtes: il s'agit de décomposer la requête en identifiant les différents ensemble de sources.
3. Identification des dépendances et création de l'arbre de dépendance: il s'agit de déterminer les dépendances entre les différents atomes de la requête.
4. Identification des sources: à l'aide de cache de description de métadonnées, on peut associer les adaptateurs correspondant au type de source demandé.
5. Création du plan d'exécution: l'arbre de dépendance est ensuite utilisé afin de générer des plans d'exécution équivalents.
6. Optimisation du plan d'exécution: il s'agit de choisir le plan d'exécution de moindre coût.

Nous allons détailler le principe et les algorithmes utilisés pour chaque étape citée ci-dessus. Afin de faciliter la compréhension de chacune des étapes, nous adoptons les notions suivantes: les lettres minuscules *x, y, z* désignent des variables XQuery, et les lettres capitales E, C, R désignent des requêtes XQuery. Pour raison de simplicité, on

abrége des requêtes de la forme: « **for** x_1 **in** E_1, x_2 **in** E_2,…, x_n **in** E_n » en « **for** X **in** E ».

Les clauses **let** peuvent être considérées comme des variables temporaires de définitions. Lors de la normalisation elles sont éliminées par la règle suivante:

for X **in** E_1 **let** Y = E_2 (X) **for** Z **in** E_3 (X , Y) **where** C (X , Y, Z) **return** R (X , Y, Z)	⇔	**for** X **in** E_1, Z **in** E_3 (X, E_2(X)) **where** C (X, E_2(X), Z) **return** R (X, E_2(X), Z)

3.3.1. Normalisation et canonisation des requêtes

La normalisation d'une requête consiste à supprimer autant que possible les imbrications. La procédure de canonisation consiste à séparer les expressions de la requête de la reconstruction. Pour cela, on sépare les blocs de requêtes XQuery en plusieurs sous-requêtes sans tenir compte de la reconstruction que l'on nommera QDB et une requête de reconstruction appelée QMem. Cette décomposition est effectuée suivant les règles ci-dessous:

for X **in** E_1 **where** C_1 (X) **return** R (X)	=>	**let** t_1 := **for** V_1 **in** E_1 **where** C1 (V_1) **return** (V_1) — QDB$_1$ R (V_1) — QMem

où $V_1 = X$

et

for X **in** E_1 **where** C_1 (X) **return** R(**for** Y in E2 **where** C2(X, Y) **return** R(X, Y))	=>	**let** t_1 := **for** V_1 **in** E_1 **where** C1 (V_1) **return** (V_1) — QDB$_2$ **let** t_2 := **for** V_2 **in** t_1 **for** V_3 **in** E_2 **where** $C_2(V_2, V_3)$ **return** (V_2, V_3) — QDB$_2$ R(V_1, V_2, V_3) — QMem

Où $\begin{cases} V_1 = X \\ V_2 = X \cap Y \\ V_3 = Y - X \end{cases}$

3.3.2. Atomisation des requêtes

Une fois la requête canonisée, il s'agit ensuite de déterminer pour chacune des sous requêtes QDB, les différents ensembles de documents. Ces différents ensembles identifiés

sont appelés requête atomique et sont notés QA_i. Et une requête de liaison notée QDEP. La décomposition en requête atomique se fait suivant la règle suivante:

for $\overrightarrow{x}$ **in** E_1 **Where** $C(\overrightarrow{x})$ **Return** $(\overrightarrow{x})$	=>	**For** $\overrightarrow{x_1}$ **in** $E_1(\overrightarrow{x_1})$ **Where** $C_{x1}(\overrightarrow{x_1})$ **Return** $(\overrightarrow{x_1})$	QA_1
		For $\overrightarrow{x_2}$ **in** $E_2(\overrightarrow{x_2})$ **Where** $C_{x2}(\overrightarrow{x_2})$ **Return** $(\overrightarrow{x_2})$	QA_2
		…..	
		for $\overrightarrow{x_n}$ **in** $E_n(\overrightarrow{x_n})$ **where** $C(\overrightarrow{x_n})$ **return** $(\overrightarrow{x_n})$	QA_n
		where $C(\overrightarrow{x_1}, \overrightarrow{x_2}, \ldots, \overrightarrow{x_n})$	$QDEP_1$

3.3.3. Identification des sources

Après l'atomisation, chaque requête est ensuite analysée afin de repérer les sources intervenant dans la requête. On utilise l'accès au cache de descriptions de métadonnées pour étendre les chemins.

```
Fonction trouverSource (requête atomique)
  Nom_collection ← recupererNomDeLaCollection (requête atomique)
  Tableau_tous_chemin[] ← résoudreTousLesCheminAbsolusPossibles (
                                                    Requête atomique)
  modifierRequête (tableau_tous_chemin[]);
 tableau_sources[] ← chercherSourceParNomDeCollection (nom_collection)
pour chaque source_i de tableau_sources[]
   vérifier que tableau_tous_chemin[] est inclus dans les chemin de source_i
   ajouterDans (tableau_sources_valide[], source_i)
fin pour
fin fonction
```

3.3.4. Création du plan d'exécution

Un plan d'exécution est une structure arborescente comportant au niveau des feuilles les sources à interroger et au niveau des nœuds, des opérateurs bien connus du monde relationnel: jointure, restriction, projection, etc. ainsi que des opérateurs de fonction. À titre d'exemple, L'arbre d'exécution d'une requête est donné à la figure 3.6.

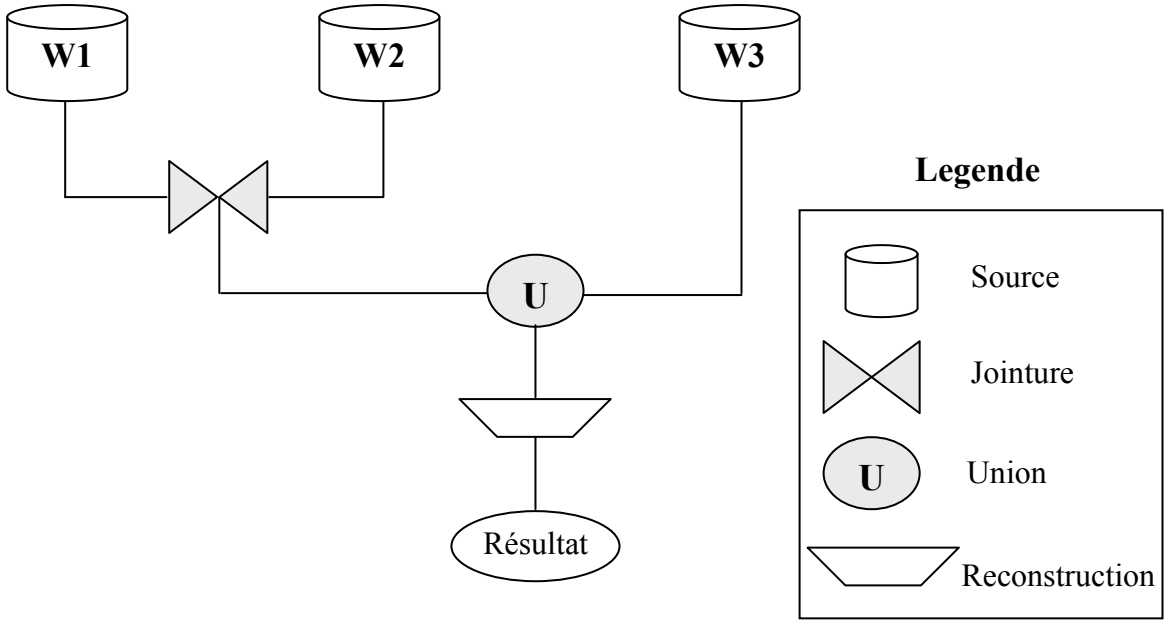


Figure 3. 6 - Plan d'exécution d'une requête

Pour chaque requête atomique QA_i s'appliquant à une source physique, une méthode de l'interface WSDL du service web, nommé ExecQA(QA_i), est exécutée. Son rôle consiste à interroger directement l'adaptateur gérant cette source avec la requête atomique associée, et de récupérer le résultat sous forme XML. Les requêtes de liaison QDep permettent de construire les opérateurs de jointure (XJoint). En suite, l'opérateur d'union (XUnion) permet de faire l'union entre le résultat de la jointure et les données issues de la source W3. Enfin, la requête QMem de reconstruction est utilisée pour construire l'opérateur de reconstruction.

3.3.5. Optimisation du plan d'exécution

À l'issu des étapes précédentes, nous obtenons un arbre algébrique permettant de définir le plan d'exécution. Ce plan d'exécution peut être optimisé par des règles de réécriture d'expressions d'opérateurs. Des règles simple et classique comme la remontée de restrictions et des projections peuvent être appliquées. Dans le cadre d'une architecture de médiation, une optimisation à la base d'ordonnancement de jointures suivant les statistiques sur les collections, ainsi que des regroupement de sous-requêtes portant sur la même source peuvent être ajoutée. Toutes ces techniques d'optimisation manipulant un arbre algébrique sont possibles.

3.3.6. Recomposition

Le recomposeur transforme la structure interne des résultats en arbre en suivant la définition de construction donnée dans la requête. Pour cela, le recomposeur remplace les variables de la QMem par les valeurs trouvées au fur et à mesure de l'évaluation. Un mécanisme de balisage créé la structure finale au fur et à mesure et insère les valeurs des variables résolues.

4.1. INTRODUCTION

Pour valider notre médiateur, nous nous sommes appuyés sur un système hétérogène expérimental composé d'un ensemble de données hétérogènes et d'une série de requêtes associées à ces données. Dans ce cadre, nous avons développé un médiateur qui n'intègre pas encore toutes les fonctionnalités décrites dans ce mémoire, en particulier les caches, les modèles de coût, et les langages d'exportation de coût et de capacité. Le seul critère étant si oui ou non, le médiateur est capable de répondre à une requête qui lui est posée.

4.2. SYSTEME HETEROGENE EXPERIMENTAL

Le schéma utilisé par ce système comporte 5 sources de données ayant des relations diverses entre elles. Trois de ces sources (1, 2 et 3) sont relationnelles, et les 2 autres (4 et 5) sont semi-structurées. La figure 4.1 montre ces sources, ainsi les relations entre elles.

Remarque : On a supposé que les 5 sources se trouvent dans des sites différents dans un environnement distribué (Internet ou intranet), et que chaque source est autonome, afin de réaliser un système de médiation supportant les trois dimensions fondamentales de l'intégration vues dans le début de chapitre 2 (l'autonomie, l'hétérogénéité et la distribution).

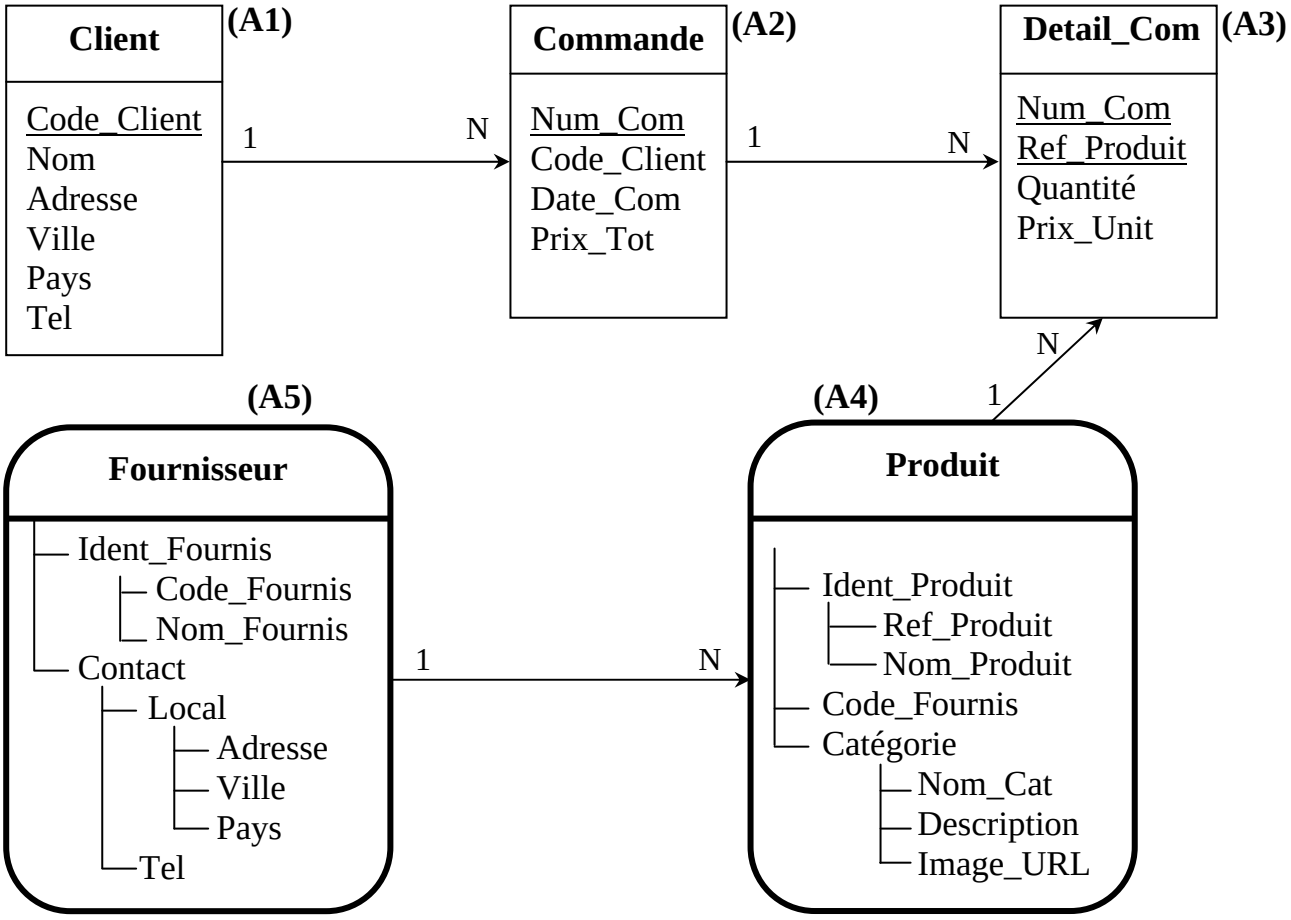


Figure 4. 1 - Schéma de données

Les schémas des tables et les schémas XML utilisés sont définis comme suit:

a) Les sources relationnelles

- **Client** (*Code_Client*: Text (5); *Nom*: Text(25); *Adresse*: text(40); Ville: Text (25); Pays: text (25) ; Tel : Text (24));
- **Commande** (*Num_Com*: Numérique; *Code_Client*: Text (5); *Date_Com*: date; *Prix_Tot*: Numérique);
- **Detail_Com** (*Num_Com*: Numérique; Ref_*Produit*: Text (7); *Quantité*: Numérique; *Prix_Unit*: Numérique).

b) Les sources semi-structurées

- **Schéma XML de la source (A4): Produit**

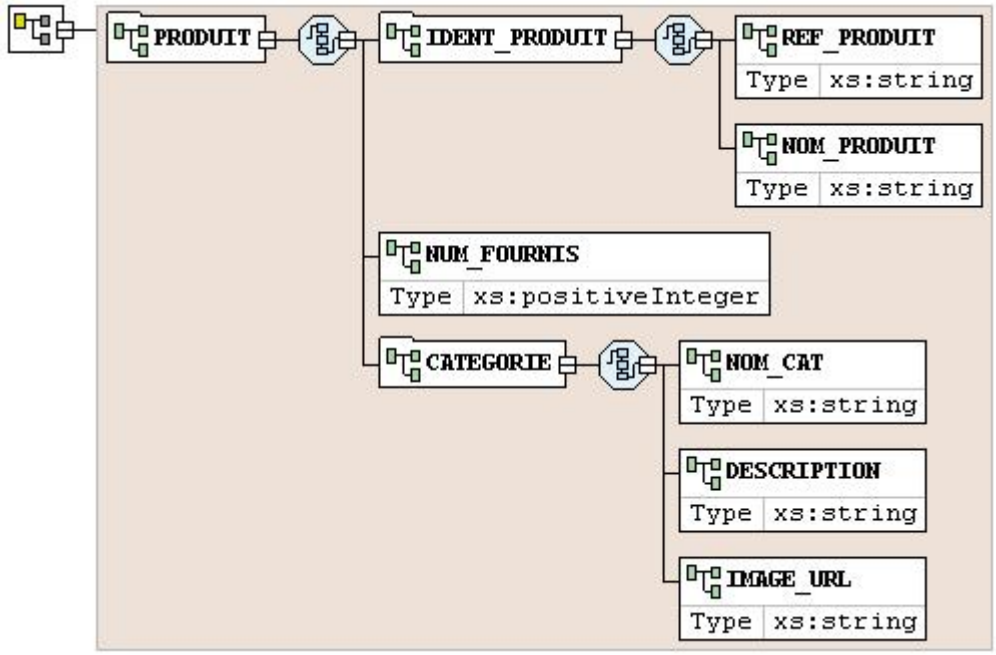
- Fichier PRODUIT.XSD

```
<?xml version="1.0"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema">

  <xs:element name="PRODUIT">
    <xs:complexType>
      <xs:sequence>
        <xs:element name="IDENT_PRODUIT">
          <xs:complexType>
            <xs:sequence>
              <xs:element name="REF_PRODUIT" type="xs:string"/>
              <xs:element name="NOM_PRODUIT" type="xs:string"/>
            </xs:sequence>
          </xs:complexType>
        </xs:element>

        <xs:element name="NUM_FOURNIS" type="xs:positiveInteger"/>
        <xs:element name="CATEGORIE">
          <xs:complexType>
            <xs:sequence>
              <xs:element name="NOM_CAT" type="xs:string"/>
              <xs:element name="DESCRIPTION" type="xs:string"/>
              <xs:element name="IMAGE_URL" type="xs:string"/>
            </xs:sequence>
          </xs:complexType>
        </xs:element>
      </xs:sequence>
    </xs:complexType>
  </xs:element>
</xs:schema>
```

- Diagramme du schéma PRODUIT



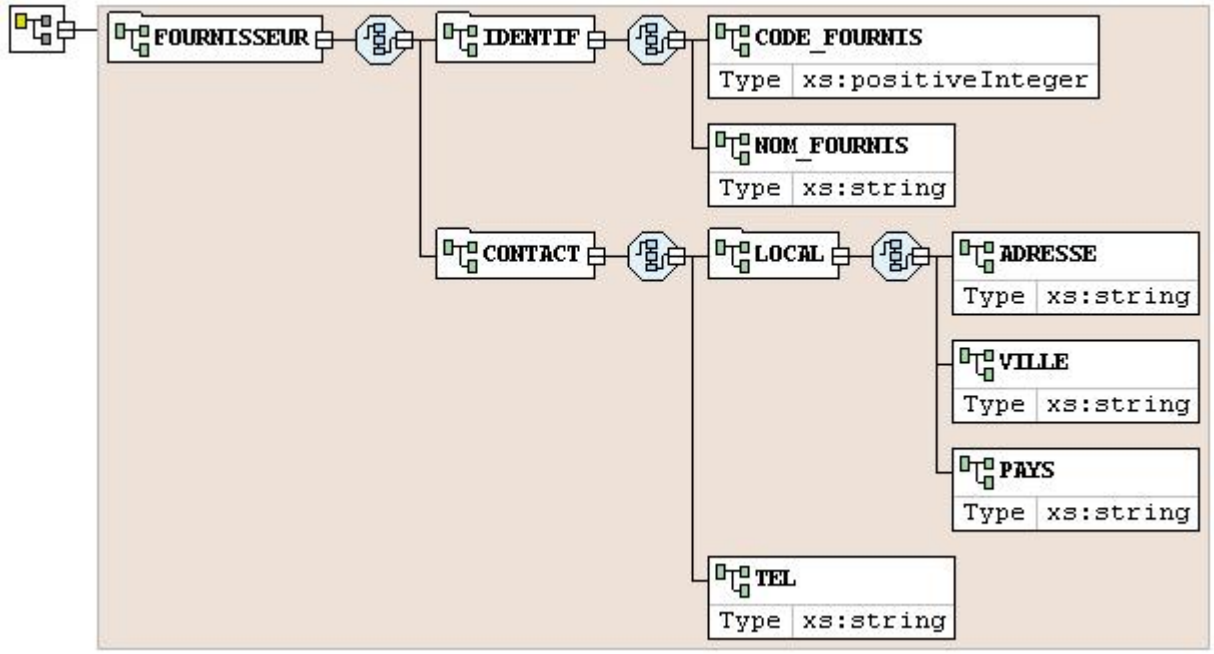
- **Schéma XML de la source (A5): Fournisseur**

- Fichier FOURNISSEUR.XSD

```
<?xml version="1.0"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema">
  <xs:element name="FOURNISSEUR">
    <xs:complexType>
      <xs:sequence>
        <xs:element name="IDENTIF">
          <xs:complexType>
            <xs:sequence>
              <xs:element name="CODE_FOURNIS"
                type="xs:positiveInteger" nillable="false">
              </xs:element>
              <xs:element name="NOM_FOURNIS" type="xs:string">
              </xs:element>
            </xs:sequence>
          </xs:complexType>
        </xs:element>
        <xs:element name="CONTACT">
          <xs:complexType>
            <xs:sequence>
```

```
                    <xs:element name="LOCAL">
                      <xs:complexType>
                        <xs:sequence>
                          <xs:element name="ADRESSE" type="xs:string"/>
                          <xs:element name="VILLE" type="xs:string"/>
                          <xs:element name="PAYS" type="xs:string"/>
                        </xs:sequence>
                      </xs:complexType>
                    </xs:element>
                    <xs:element name="TEL" type="xs:string"/>
                  </xs:sequence>
                </xs:complexType>
              </xs:element>
            </xs:sequence>
          </xs:complexType>
        </xs:element>
      </xs:schema>
```

- Diagramme du schéma FOURNISEUR



4.2.1. Médiateur

Afin de pouvoir classifier différents cas d'intégration de sources, nous définissons deux architectures de médiation. L'architecture (a) accède directement à toutes les sources, et l'architecture (b) accède à des médiateur qui eux-mêmes intègrent des sources de données réparties. Les deux architectures de médiation sont illustrées à la figure 4.2. Pour notre exemple nous avons utilisé l'architecture (a), à cause de sa simplicité et qu'elle intègre les sources relationnelles et les sources XML.

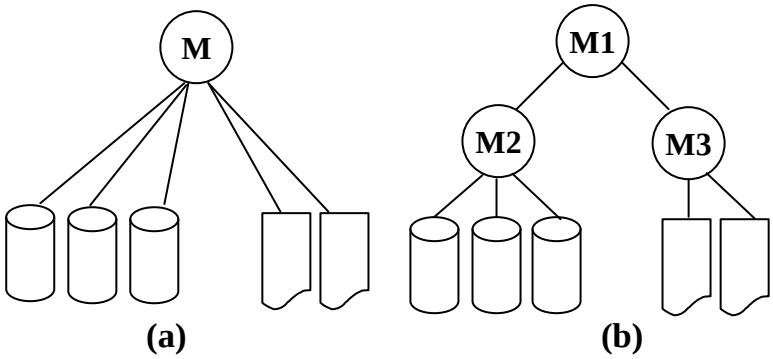


Figure 4. 2 - Différentes architectures de médiation

Pour connaître la localisation des différentes sources, le médiateur possède un fichier de configuration représenté par le document XML suivant :

```
<mediateur name="M">
        <adaptateur name="A1">
              <connexion>
                  http://localhost/sources/Client_service.wsdl
             </connexion>
        </adaptateur>
        <adaptateur name="A2">
              <connexion>
                  http://localhost/sources/Commande_service. wsdl
             </connexion>
        </adaptateur>
        <adaptateur name="A3">
              <connexion>
                  http://localhost/sources/Detailcom_service. wsdl
             </connexion>
        </adaptateur>
        <adaptateur name="A4">
              <connexion>
                  http://localhost/sources/Produit_service. wsdl
             </connexion>
        </adaptateur>
        <adaptateur name="A5">
              <connexion>
                  http://localhost/sources/Fournisseur_service. wsdl
             </connexion>
        </adaptateur>
</mediateur>
```

Chaque adaptateur est localisé sur le réseau par une description WSDL du service Web, indiquée par une adresse URL dans l'élément 'connexion'.

4.2.2. Adaptateur

Les sources (A1, A2 et A3) sont des sources relationnelles. Elles sont chacune pourvues d'un adaptateur permettant d'interroger une base de données relationnelles et de renvoyer les résultats sous forme XML. (A4 et A5) sont des sources de données semi-structurées, elles sont chacune représentée par un document XML. Chaque adaptateur offre un service web représenté par une interface WSDL et accessible via le protocole SOAP. La figure 2.3 présente le fichier WSDL d'un adaptateur.

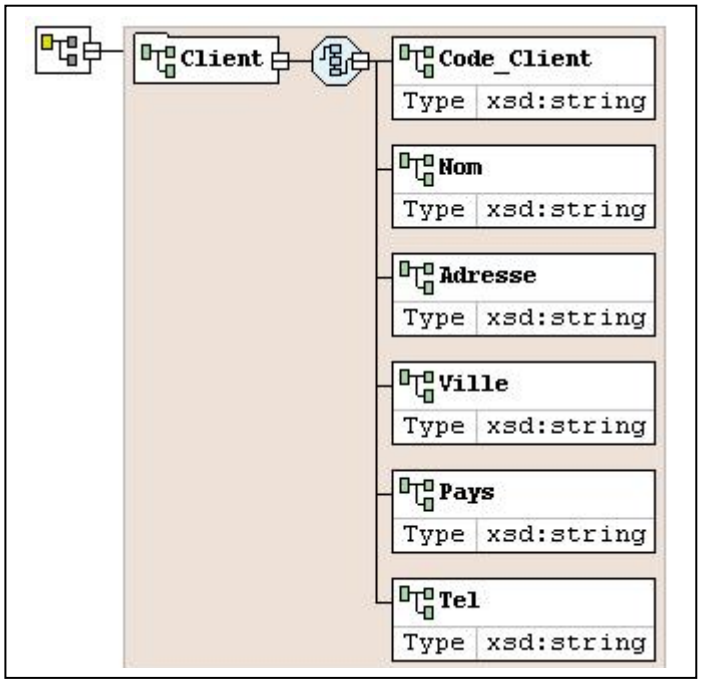
Lors de l'initialisation, le médiateur demande aux adaptateurs (avec la méthode *getShemas( )*) de lui communiquer les schémas des sources de données, permettant de définir les décisions à prendre pour l'évaluation d'une requête. Via le protocole SOAP, le médiateur invoque la méthode *getShemas( )* du service Web fourni par chaque adaptateur. L'invocation et la réponse sont codées comme suit :

```
<?xml version="1.0"?>
<SOAP-ENV:Envelope xmlns:SOAP-ENV="http://www.w3.org/2001/09/soap-envelope/">
   <SOAP-ENV:Body>
      <s:getSchemas xmlns:s="URL de la source"/>
   </SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```

La réponse sera codée comme suit :

```
<?xml version="1.0"?>
<SOAP-ENV:Envelope xmlns:SOAP-ENV="http://www.w3.org/2001/09/soap-envelope/">
   <SOAP-ENV:Body>
      <s:SchemasReturn xmlns:s="URL de la source"/>
          <Reponse>Schéma de la source</Reponse>
      </s:SchemasReturn>
   </SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```

Concernant les sources relationnelles, leurs schémas seront convertis en schémas XML. Par exemple: le schéma XML correspondant au schéma de la table 'Client' après transformation est présenté à la figure ci-contre.



Enfin, le médiateur encapsule tous les schémas reçus dans un seul schéma, appelé schéma global, qui est représenté dans La figure 4.3.

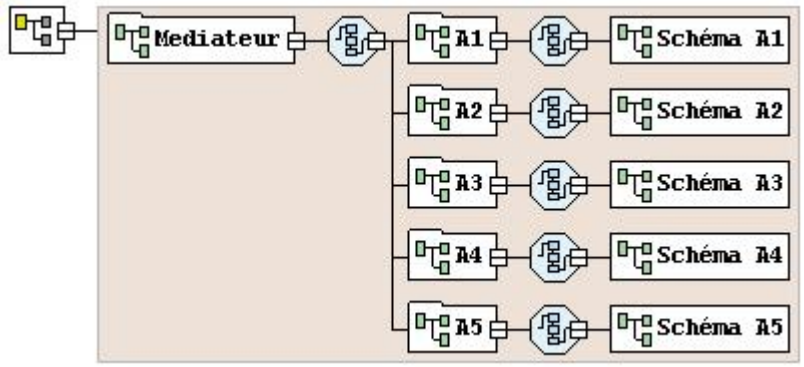


Figure 4. 3 - Schéma global du médiateur

4.2.3. L'échange de données Médiateur / adaptateur

L'échange de requêtes entre médiateur et adaptateur pourra s'appuyer sur SOAP, bien qu'un protocole plus compact et moins lourd puisse être plus approprié pour obtenir de meilleures performances. La figure 4.4 illustre l'interaction entre médiateur et adaptateur.

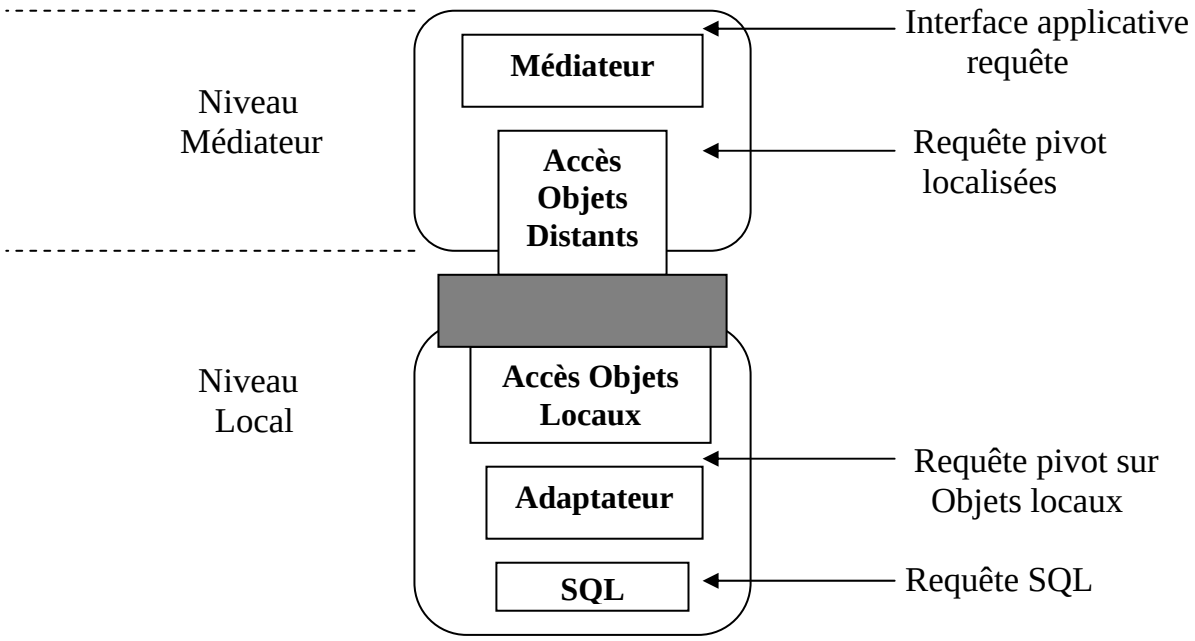


Figure 4. 4 - Interaction entre médiateur et adaptateur

4.2.4. Modèle de données et types de requête

Dans notre architecture, nous faisons appel aux données de type relationnelle et aux données semi structurées. Celles-ci sont dispersées sur différentes sources et les attributs de ces différentes données sont générés de sorte à pouvoir réaliser des jointures inter-sites. Les

requêtes que nous utiliserons sont des requêtes XQuery. Nous noterons $Q(A_1),\ldots,Q(A_5)$ des sous-requêtes faisant intervenir les adaptateurs $A_1, \ldots, A_5$.

4.3. PROCESSUS D'INTEGRATION

L'objectif du médiateur et de réaliser la fédération de sources de données hétérogènes en XML et de permettre l'interrogation des sources fédérées avec XQuery, le nouveau standard d'interrogation de données. Le médiateur vise à donner une vue unique d'un ensemble de sources sous forme d'une collection de documents XML. La figure 4.5 illustre les fonctionnalités essentielles:

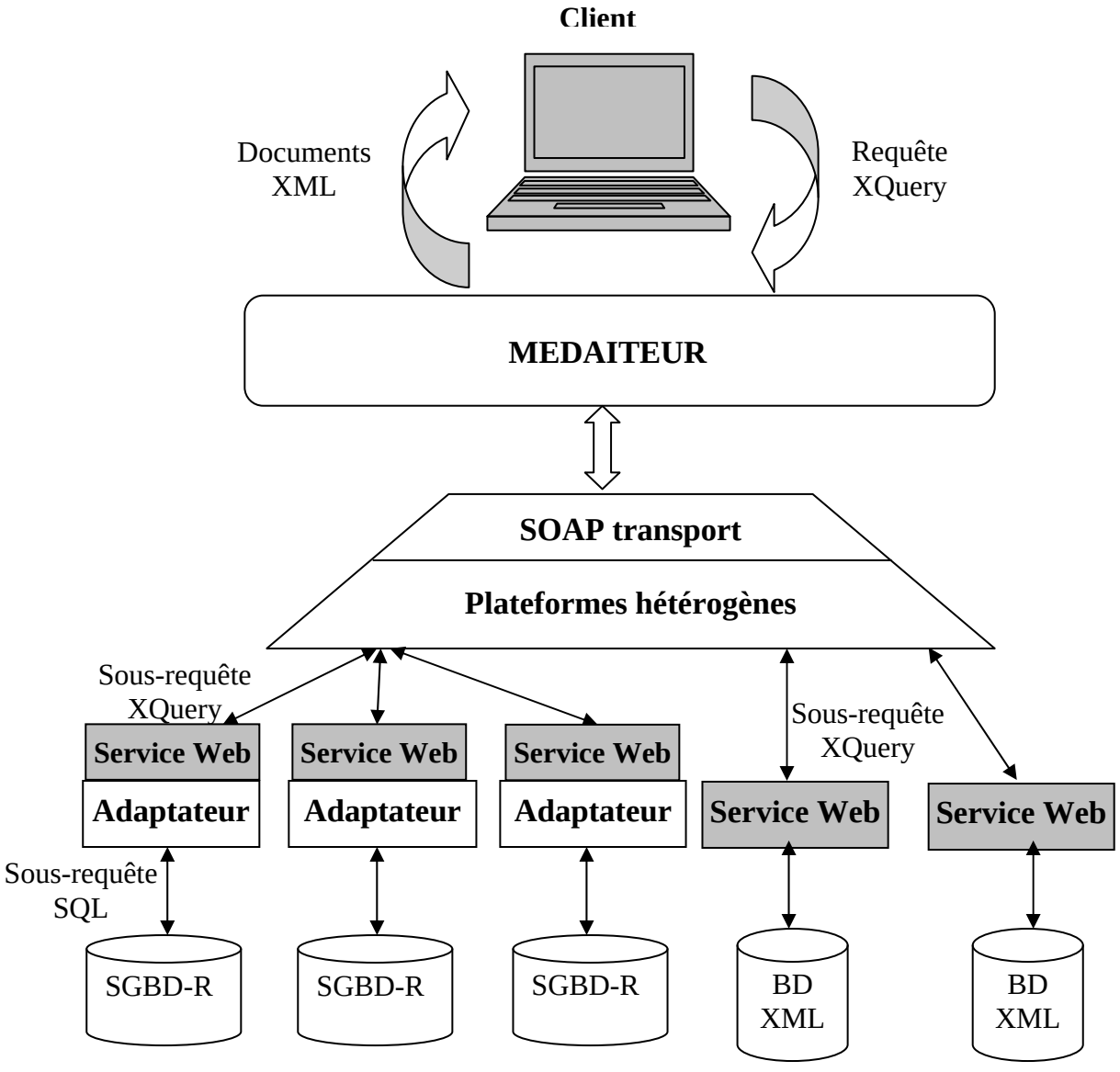


Figure 4. 5 - Processus d'intégration

Le médiateur reçoit des requêtes XQuery qu'il décompose en sous-requêtes, il délègue aux adaptateurs les sous-requêtes sous forme de XQuery plus ou moins complexes selon la capacité des sources.

Chaque adaptateur local exécute la sous-requête et retourne un document XML. Les documents issus des sources sont fusionnés et joints à la volée au sein du médiateur pour terminer l'évaluation de la requête et composer le document XML final délivré à l'utilisateur sous forme HTML pour l'afficher sur écran en utilisant les feuilles de styles XSL.

4.3.1. Requête exemple XQuery

Pour chacune des étapes suivantes, nous allons dérouler au fur et à mesure le traitement appliqué à l'évaluation de la requête exemple donnée ci-dessous :

Q: « Renvoyer tous les clients faisant une commande de l'article "Imprimante" fourni par le fournisseur "BISYS" ».

```
for $C in collection ("Client") / Client
for $M in collection ("Commande") / Commande
where
        $C / Code_Client = $M / Code_Client
return
        <CLIENT>
            <Nom_Client> $C / Nom </Nom_Client>
             <Pays> $C / Pays </Pays>
             <Produits>
                    for $D in collection ("Detail_Com") / Detail_Com
                    for $P in collection ("Produit") / Produit
                    for $F in collection ("Fournisseur") / Fournisseur
                    where
                            $M / Num_Com = $D / Num_Com                          and
                            $D / Ref_Produit = $P / Ident_Produit/Ref_Produit     and
                            $P / Code_Fournis = $F /Ident_Fournis/ Code_Fournis   and
                            $P /Ident_Produit/ Nom_Produit = "Imprimante"         and
                            $F / Ident_Fournis / Nom_Fournis = " BISYS"
                  return
                    <Produit>
                        <Nom_Produit> $P/Ident_Produit/Nom_Produit </Nom_Produit>
                        <Prix_Unitaire> $D/Prix_Unit</Prix_Unitaire>
                        <Image> $P/Categorie/Image_URL</Image>
                   </Produit>
            </Produits>
        </ CLIENT >
```

4.3.2. Normalisation et canonisation de la requête

La requête précédente interroge toutes les sources du schéma de la figure 4.1. Pour rendre cette requête sous une forme canonique, on applique l'algorithme décrit dans le chapitre précédent:

let *t1* := **for** $C **in** collection ("Client") / Client **for** $M **in** collection ("Commande") / Commande **where** $C/Code_Client = $M/Code_Client **return** ($C/Nom, $C/Pays, $M/Code_Client, $M/Num_Com)	**QDB1**
let *t2* := **for** $*t* **in** $*t1* **for** $D **in** collection ("Detail_Com") / Detail_Com **for** $P **in** collection ("Produit") / Produit **for** $F **in** collection ("Fournisseur") / Fournisseur **where** $*t* / Num_Com = $D / Num_Com **and** $D / Ref_Produit = $P / Ident_Produit/Ref_Produit **and** $P / Code_Fournis = $F /Ident_Fournis/ Code_Fournis **and** $P /Ident_Produit/ Nom_Produit = "Imprimante" **and** $F / Ident_Fournis / Nom_Fournis = " BISYS" **return** ($P /Ident_Produit/ Nom_Produit, $D/Prix_Unit, $P/Categorie/Image_URL)	**QDB2**
<CLIENT> <Nom_Client> $C / Nom </Nom_Client> <Pays> $C / Pays </Pays> <Produits> <Produit> <Nom_Produit>$P/Ident_Produit/Nom_Produit </Nom_Produit> <Prix_Unitaire> $D/Prix_Unit</Prix_Unitaire> <Image> $P/Categorie/Image_URL</Image> </Produit> </Produits> </ CLIENT >	**QMem**

QDB1 et QDB2 sont des bloques de requêtes canoniques, QMem sert à mémoriser la reconstruction du résultat de la requête t, t1, t2 sont des variables temporaires.

4.3.3. Atomisation des requêtes

Elle s'agit de déterminer dans chacune des requêtes QDB, les différents ensembles de collections de documents.

La requête QDB1 comporte deux ensembles de collections, sont appelées requêtes atomiques et sont notées QA1, QA2 et une requête de liaison QL1.

for $C **in** collection ("Client") / Client **return** ($C/Nom, $C/Pays)	**QA1**
for $M **in** collection ("Commande") / Commande **return** ($M/Code_Client, $M/Num_Com)	**QA2**
$C/Code_Client = $M/Code_Client	**QL1**

La requête QDB2 comporte 3 requêtes atomiques QA3, QA4, QA5, et une requête de liaison QL2.

for $D **in** collection ("Detail_Com") / Detail_Com **return** ($D/Prix_Unit)	**QA3**
for $P **in** collection ("Produit") / Produit **where** $P /Ident_Produit/ Nom_Produit = "Imprimante" **return** ($P /Ident_Produit/ Nom_Produit, $P/Categorie/Image_URL)	**QA4**
for $F **in** collection ("Fournisseur") / Fournisseur **where** $F / Ident_Fournis / Nom_Fournis = " BISYS"	**QA5**
$*t* / Num_Com = $D / Num_Com **and** $D / Ref_Produit = $P / Ident_Produit/Ref_Produit **and** $P / Code_Fournis = $F /Ident_Fournis/ Code_Fournis	**QL2**

La figure ci-dessous illustre la requête exemple ainsi les requête atomiques issues des deux étapes précédentes (canonisation et atomisation) de notre médiateur.

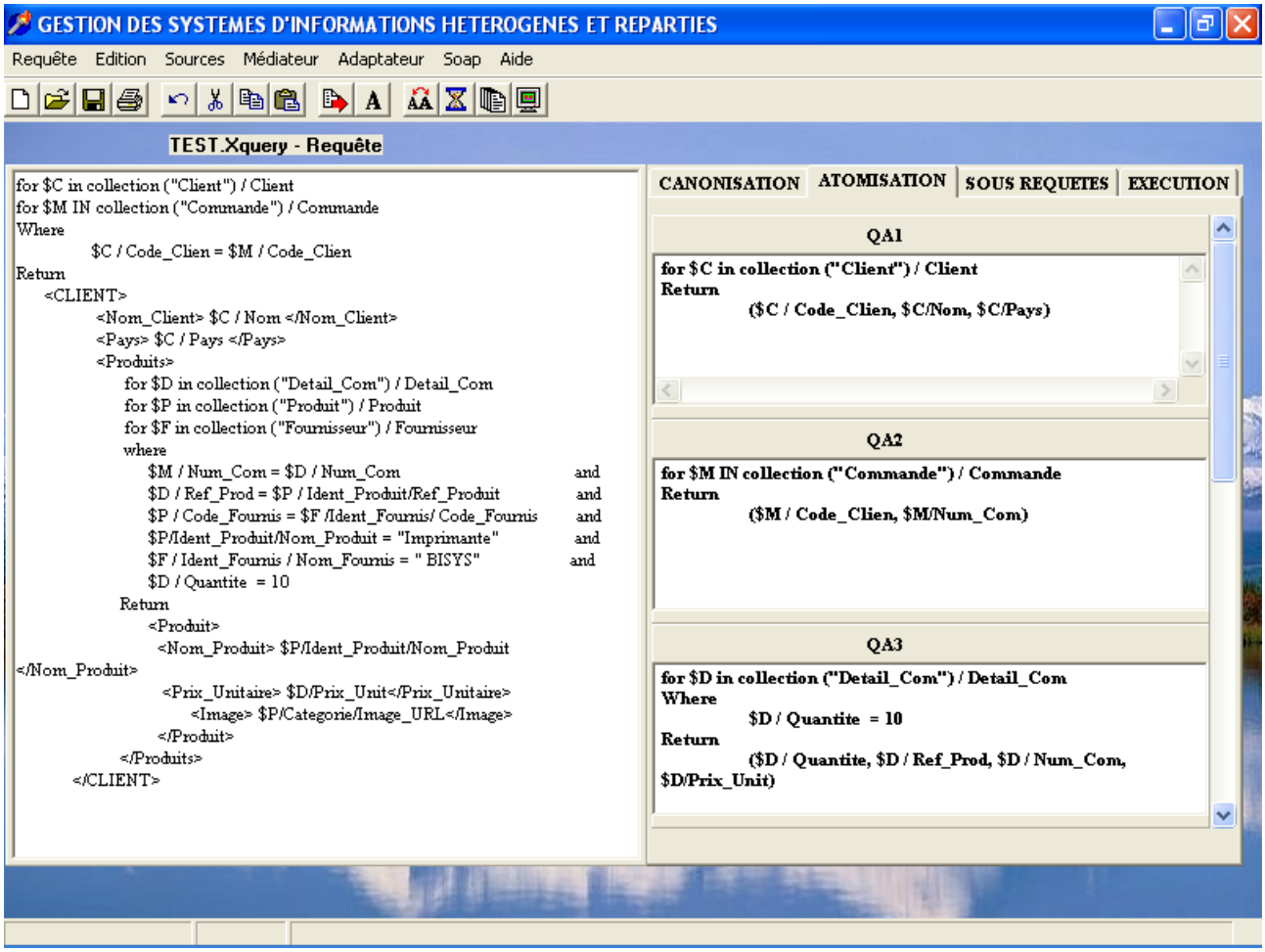


Figure 4. 6 - La requête globale et les requêtes atomiques

4.3.4. Identification des sources

Chaque requête est analysée afin de repérer les sources intervenants dans la requête. Pour cela, on utilise le schéma global représenté dans la figure 4.2.

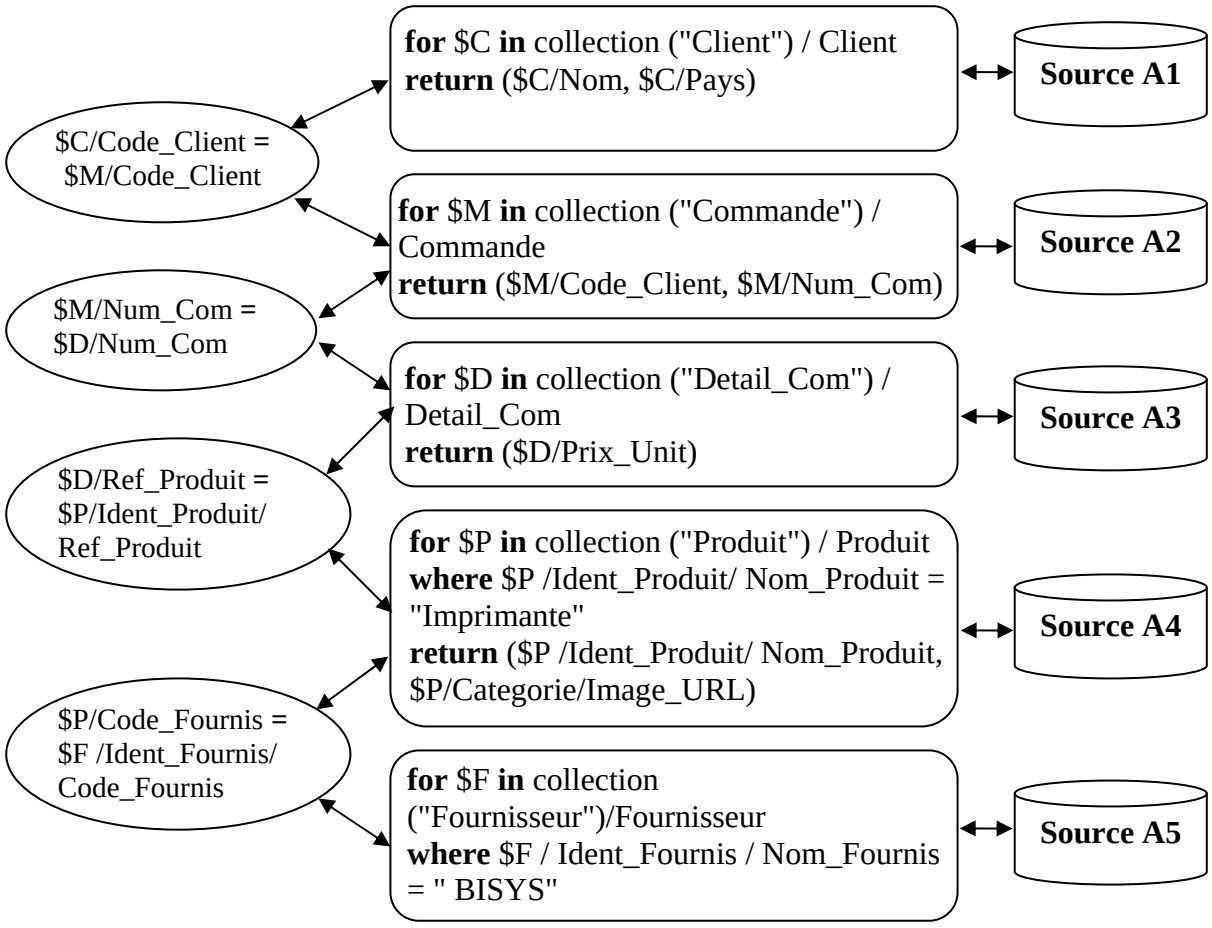


Figure 4. 7 - Identification des sources.

4.3.5. Décomposition des requêtes

Consiste à reformuler les 5 sous requêtes précédentes en éliminant les jointures inter sources. Les 5 sous requêtes sont présentées ci-dessous :

- **Q1:** Sous-requête destinée à la source A1:

```
For $C in collection("Client") / client
Return
        <Client>
              <Code_Client> $C / C_Client  </ Code_Client >
              <Nom_Client> $C / Nom  </Nom_Client>
              <Pays> $C/Pays </Pays>
        </Client>
```

- **Q2:** Sous-requête destinée à la source A2:

```
For $M in collection("Commande") / Commande
Return
        < Commande >
            <Num_Com> $M / Num_Com </ Num_Com >
            <Code> $M / Code_Client < / Code >
        </ Commande >
```

- **Q3:** Sous-requête destinée à la source A3:

```
for $D in collection ("Detail_Com") / Detail_Com
Return
        < Detail_Com >
            <Num_Com > $D/Num_Com </ Num_Com >
            <Ref_Produit > $D/Ref_Produit < Ref_Produit >
            <Prix_Unit> $D/Prix_Unit < / Prix_Unit >
        </ Detail_Com >
```

- **Q4:** Sous-requête destinée à la source A4:

```
for $P in collection ("Produit") / Produit
where
        $P /Ident_Produit/ Nom_Produit = "Imprimante"
Return
        < Produit >
            < Ref_Produit > $P/Ident_Produit/Ref_Produit </ Ref_Produit >
            < Code_Fournis > $P/Code_Fournis < / Code_Fournis >
            <Image> $P/Categorie/Image_URL </ Image >
        </ Produit >
```

- **Q5:** Sous-requête destinée à la source A5:

```
for $F in collection ("Fournisseur")/Fournisseur
where
     $F / Ident_Fournis / Nom_Fournis = " BISYS "
Return
        < Fournisseur >
            < Code_Fournis > $F /Ident_Fournis/ Code_Fournis </ Code_Fournis >
            < Nom_Fournis > $F / Ident_Fournis / Nom_Fournis < / Nom_Fournis >
        </ Fournisseur >
```

4.3.6. L'envoi des sous requêtes

Après la décomposition de la requête et l'identification des sources intervenants dans chaque sous-requête**,** le médiateur peut envoyer les sous requêtes aux différents adaptateurs en invoquant la méthode 'ExecQuery(Qi)', où Qi présente une sous requête (Q1,Q2,...Q5).

L'invocation et la réponse sont réalisées via le protocole SOAP, et sont codées comme suit :

```
<?xml version="1.0"?>
<SOAP-ENV:Envelope xmlns:SOAP-ENV=" http://www.w3.org/2001/09/soap-envelope /">
   <SOAP-ENV:Body>
      <s:ExecQuery xmlns:s=" URL de la source ">
         <s:Q1/>
      </s:ExecQuery>
   </SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```

La réponse sera codée comme suit :

```
<?xml version="1.0"?>
<SOAP-ENV:Envelope xmlns:SOAP-ENV="http://www.w3.org/2001/09/soap-envelope/">
   <SOAP-ENV:Body>
      <s:QueryReturn xmlns:s="URL de la source"/>
          <Reponse>résultat sous forme XML</Reponse>
      </s: QueryReturn >
   </SOAP-ENV:Body>
</SOAP-ENV:Envelope>
```

4.3.7. Transformation XQuery en SQL

Les sous-requêtes Q1, Q2 et Q3 seront transformer respectivement en sous-requêtes SQL (SQL1, SQL2 et SQL3) par l'intermédiaire des adaptateurs A1, A2 et A3. Chaque adaptateur utilise un analyseur de requête XQuery, en général un traducteur qui à partir d'une requête XQuery génère des requêtes locales SQL nécessaires à l'évaluation. Et les deux sous-requêtes Q4 et Q5 ne changent pas.

SQL1 :

```
SELECT  Code_Client,  Nom, Pays
FROM  Client
```

SQL2 :

```
SELECT  Num_Com,  Code_Client
FROM  Commande
```

SQL3 :

```
SELECT  Num_Com,  Ref_Produit, Nom_Art, Prix_Unit
FROM  Detail_Com
```

4.3.8. Evaluation des requêtes

L'exécution des sous-requêtes se fait en parallèle sur les différentes sources de données (relationnelles et semi-structurées). Les documents issus des sources (après le transfert du relationnel à XML) sont fusionnés et joints à la volée au sein du médiateur pour terminer l'évaluation de la requête et composer le document XML final délivré à l'utilisateur sous forme d'une page Web.

4.3.9. Du relationnel à l'XML

La transformation du relationnel à XML consiste à produire un document de racine convenue (par exemple <TABLE> où *TABLE* est le nom de la table) composé d'un élément pour chaque tuple de balise convenue (par exemple <Tuple>) et d'élément imbriqués de premier niveau représentant chacune des attributs du tuple.

Code_Client	Nom	Adresse	Ville	Pays	Tel
C0054	abababb	aaaaa	Setif	ALG	036.15.22.90
C0126	ssssss		Setif	ALG	036.14.25.93

```
<Client>
    <Tuple>
        <Code_Client>"C0054" </Code_Client>
        <Nom> "abababb"  </Nom>
        <Adresse> "aaaaa" </Adresse>
        <Ville>"Setif"</Ville>
        <Pays>"ALG" </Pays>
        < Tel> "036.15.22.90" </Tel>
    </Tuple>
    <Tuple>
        <Code_Client>"C0126" </Code_Client>
        <Nom> "ssssss"  </Nom>
        <Ville>"Setif"</Ville>
        <Pays>"ALG" </Pays>
        < Tel> "036.14.25.93" </Tel>
    </Tuple>
```

Figure 4. 8 - Traduction du relationnel à XML

4.3.10. Recomposition des documents

A l'issu des étapes précédentes, nous obtenons des documents XML, et à partir de ces documents le médiateur utilise la requête globale pour composer l'arbre de jointure et l'opérateur d'imbrication et de produire le résultat XML. La figure 4.8 illustre le processus de fusionnement des documents.

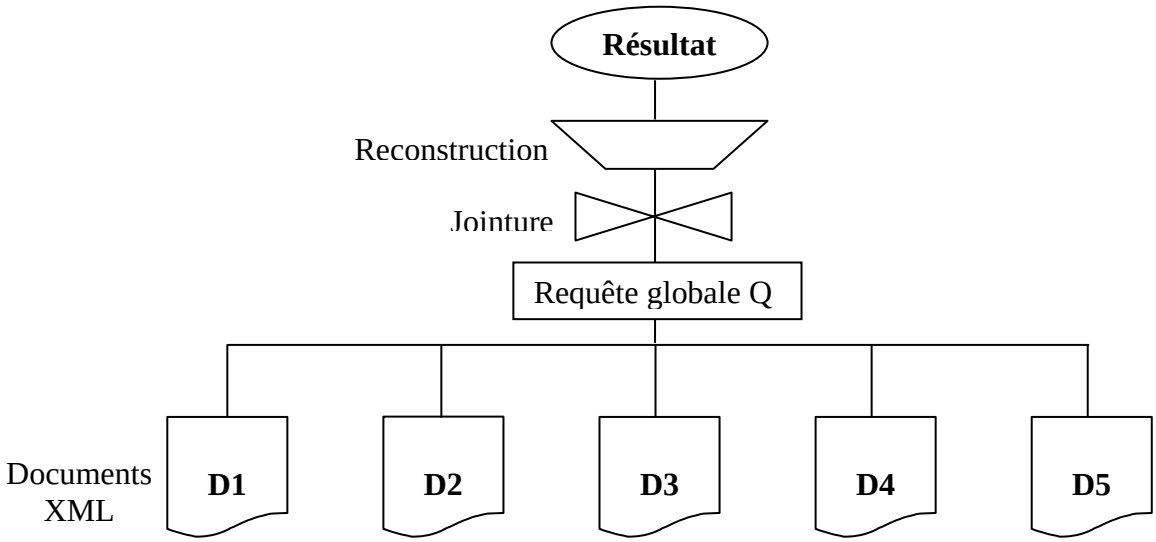


Figure 4. 9 - Fusionnement des documents XML

4.3.11. Présentation du document XML résultat

A la fin des étapes précédentes, nous obtenons un document XML résultat présenté à la figure 4.9.

```
<Clients>
    <Client>
      <Nom_Client>ababab</ Nom_Client >
      <Pays>ALG</Pays>
      <Produits>
          <Produit>
            <Nom_produit>"Imrimante"</Nom_produit>
            <Prix_Unitaire>5000</ Prix_Unitaire >
            <Image>"Image125.jpg"</Image>
          </Produit>
      </ Produits >
    </Client>
     <Client>
      <Nom_Client>samsam</ Nom_Client >
      <Pays>ALG</Pays>
      <Produits>
          <Produit>
            <Nom_produit>"Imrimante"</Nom_produit>
            <Prix_Unitaire>4500</ Prix_Unitaire >
            <Image>"Image871.jpg"</Image>
          </Produit>
      </ Produits >
    </Client>
</Clients>
```

Figure 4. 10 - Document XML résultat

Enfin, le document XML est prés à l'affichage sur écran sous forme d'une page web, à cet effet nous utilisons les feuilles de style (XSL) qui sont des fichiers associés aux documents XML contenant une séquence de commandes précisant les éléments de données à présenter et les mises en pages associées pour la publication de données sur un terminal.

Dans le principe, une définition de mise en page peut comporter des spécifications de transformation et de formatage d'objets. Ainsi, une feuille de style XSL permet de transformer un document XML d'entrée en un autre document XML. Le document XML résultant de la transformation peut être pris en charge par un outil de formatage, qui crée une version imprimable selon un format classique, PDF, DVI, RTF, HTML, ou tout autre format, (dans notre exemple nous avons utilisé le format HTML). D'où la règle de transformation suivante :

XML + XSL = HTML

La figure 4.8 illustre cette transformation.

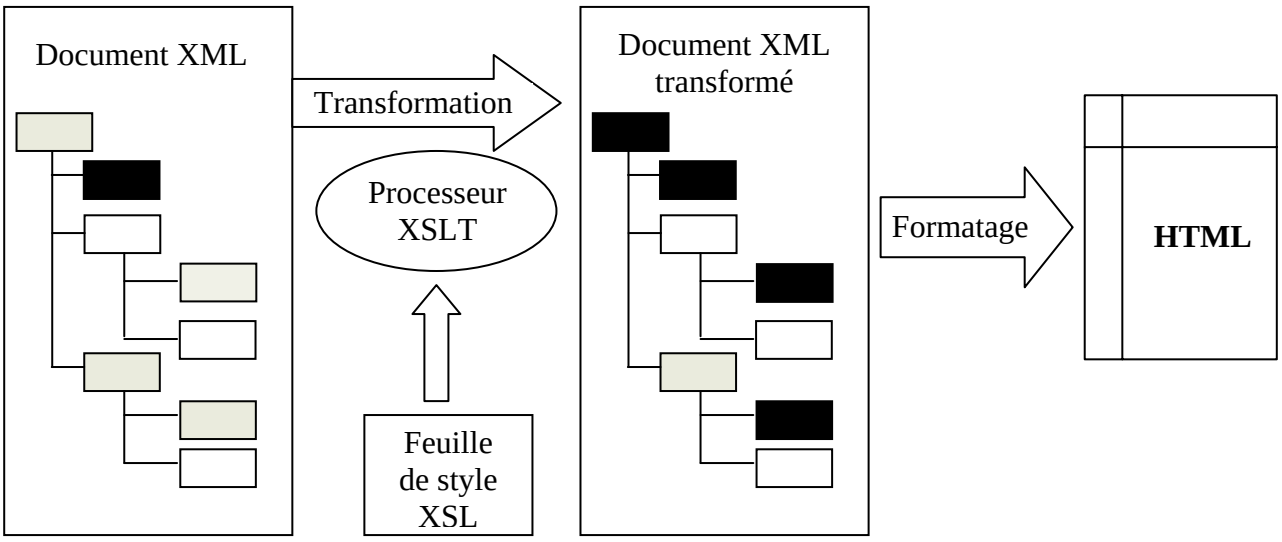


Figure 4. 11 - Transformation et formatage de document XML

CONCLUSION

Dans cette thèse, nous avons étudié les solutions à la médiation de données issues de sources hétérogènes. La médiation réalise l'intégration virtuelle de données pour répondre aux requêtes des utilisateurs qui manipulent des vues intégrées des données. Les concepts de base et une architecture de référence ont été présentés. Nous avons ensuite examiné les générations des systèmes de médiation.

Nous avons étudié les caractéristiques d'une architecture de médiation faisant intervenir des données semi-structurées comme modèle de données interne. Nous avons montré les différents concepts liés aux données semi-structurées et les relations qui existent entre eux. Ainsi que les difficultés que présentait l'évolution des travaux existants du relationnel et objets vers le semi-structurées.

Nous avons aussi étudié les mécanismes permettant d'évaluer une requête au sein du moteur d'intégration. Pour cela, nous avons étudié une algèbre XQuery permettant de modéliser les opérations à effectuer pour répondre à une requête données. Partant du fait que l'évaluation d'une requête doit être la plus efficace possible, nous nous somme penché sur les différentes manières d'optimiser une requête en proposant un modèle de coût.

Enfin, et en appuyant sur un système hétérogène expérimental, nous avons développé un médiateur qui n'intègre pas encore toutes les fonctionnalités décrites à ce mémoire, d'où les extensions et les perspectives suivantes :

- **Utilisation d'un cache sémantique**: Il se charge de conserver au fur et à mesure les résultats de requêtes antérieures dans le but de les réutiliser ultérieurement afin d'économiser des coûts de communication.
- **Utilisation du Web sémantique :** Il s'agit d'un adaptateur de page Web pour des recherches « plein texte » et des recherches sémantiques.

A.1. QUELQUES EXEMPLES DE MÉDIATEUR XML

La plupart des architectures de médiation se sont orientées vers l'architecture DARPA I3. Dans cette section, nous présentons des architecture de médiation prenant en compte les données semi structurées.

A.1.1. TSIMMIS, GARLIC et MIX

TSIMMIS (The Stanford-IBM Manager of Multiple Information Sources) [Chawathe et al. 1994] est un projet permettant de développer des outils qui facilitent l'intégration de sources d'information hétérogènes comportant des données structurées et semi structurées.

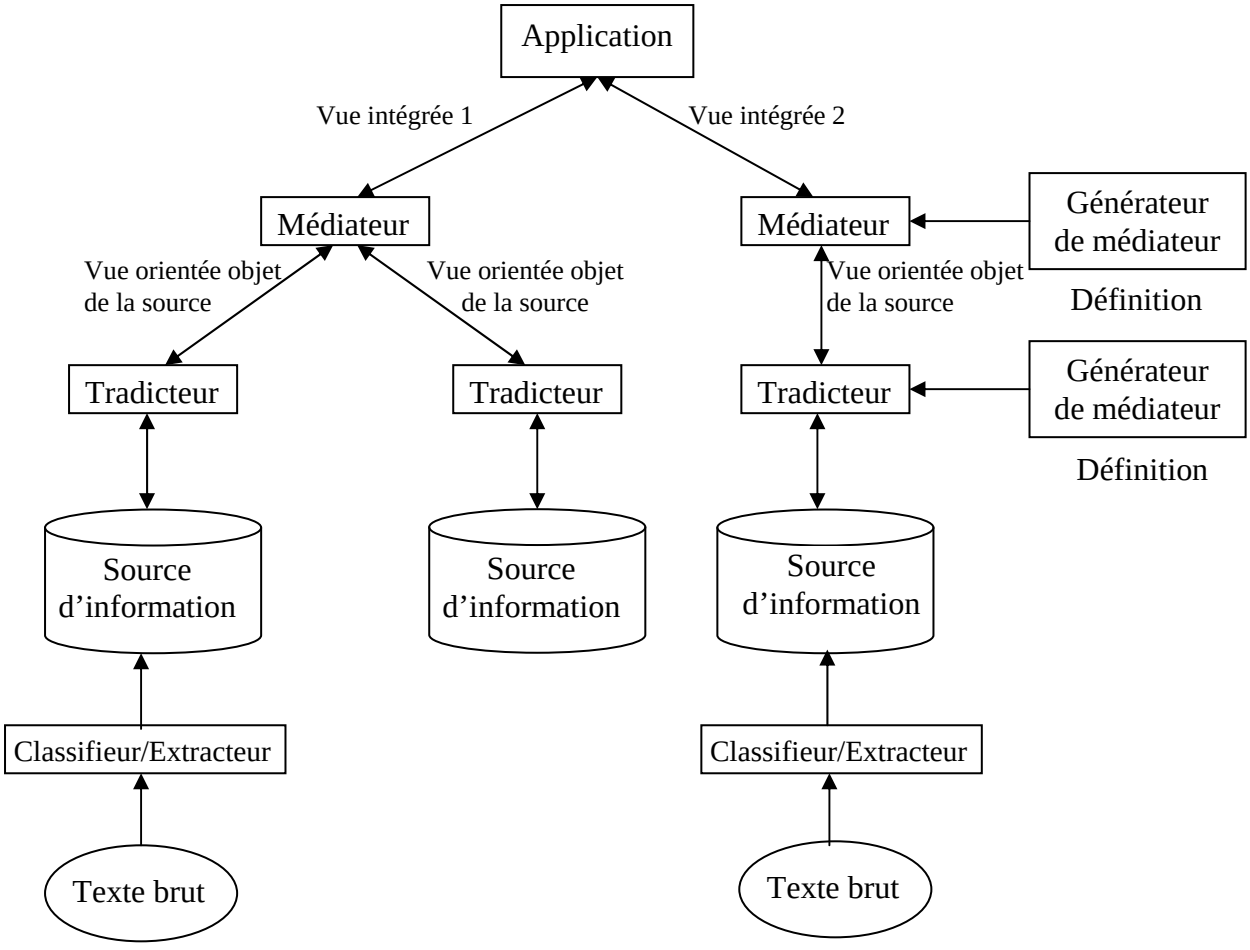


Figure A.1- Architecture de TSIMMIS

De ce fait, TSIMMIS est une architecture composé des modules suivants (voir figure A.1) :

Classifieur/ extracteur de données sur le Web : A partir du texte brut, des pages HTML, ou des pages XML, le classifieur/extracteur identifie et classifie de tels objets (ex. est-ce un fichier texte ? une page web HTML ? un courrier électronique ?), et en extrait des propriétés (ex. date, auteur).
Traducteur (ou adaptateur) : permettant la transformation des requêtes et des données en requêtes spécifiques à la source. Il convertit ensuite les données résultats de la source en données du modèle commun.
Médiateur : Permettant de combiner les informations des différentes sources.
Applications clientes : permettant de naviguer dans les données à travers le web et d'offrir une interface conviviale à l'utilisateur final.

Un des objectifs de TSIMMIS est d'intégrer des sources qui sont très hétérogènes, qui peuvent être peu structurées et qui sont susceptibles d'évoluer rapidement. TSIMMIS a introduit le formalisme OEM [Papakonstantinou et al. 1995] comme modèle de données interne et a adopté un langage de requête dérivé d'OQL : OEM-QL.

A.1.2. STRUDEL

STRUDEL [Fernandez et al. 1998] est un constructeur de sites web permettant de définir quelles sont les données qui seront disponibles sur le site. L'idée principale est de séparer l'administrateur des données du site, la création et la gestion de la structure du site, et la représentation graphique des pages web finales. Pour cela, le constructeur de site crée un modèle uniforme de toutes les données disponibles sur le site. Puis, le constructeur de site utilise ce modèle pour définir la structure du site web en appliquant aux données, une requête de « définition de site ». Enfin, le constructeur de site spécifie la représentation graphique des pages en utilisant le langage de STRUDEL pour la définition de modèle (template) HTML. Les données résident soit sur des sources externes (SGBD, fichiers structurés), soit dans l'entrepôt (*repository*) interne de STRUDEL. A chaque niveau du système STRUDEL, toutes les données (qu'elles soient internes ou externes) sont modélisées par un graphe orienté similaire à OEM. Un ensemble d'adaptateurs spécifiques se charge de convertir la représentation externe en graphe. La vue intégrée des données est appelée graphe de données.

L'architecture de STRUDEL est présentée à la figure A.2. Elle est composée de :

Entrepôt de données STRUDEL : le graphe de données d'un site web est stocké dans l'entrepôt de données STRUDEL. Les données sont obtenues des adaptateurs qui convertissent les données des sources externes en des données au format interne semi-structuré utilisé par STRUDEL.
Navigateur/editeur de graphe : il permet à l'utilisateur de créer, mettre à jour et visualiser les graphes pouvant être utilisés pour le graphe de données et le graphe du site.
Médiateur : il fourni une vue uniforme des données sous-jacentes. Plutôt que d'interroger les sources externes à la demande au moment de l'exécution de la requête,

son approche est d'intégrer les données en stockant préalablement les données des sources externes dans l'entrepôt de données STRUDEL.
Processeur de requête : STRUDEL définit le langage STRUQL (STRUdel Query Language) pour réaliser des requêtes et restructurer des données semi-structurées. L'interpréteur de requêtes de STRUDEL utilise les opérateurs physiques traditionnels (jointure, restriction, sélection) ainsi que les opérateurs nécessaires pour l'interrogation de schéma (ex. trouver tous les noms d'attributs dans un graphe).
Générateur HTML : pour 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.

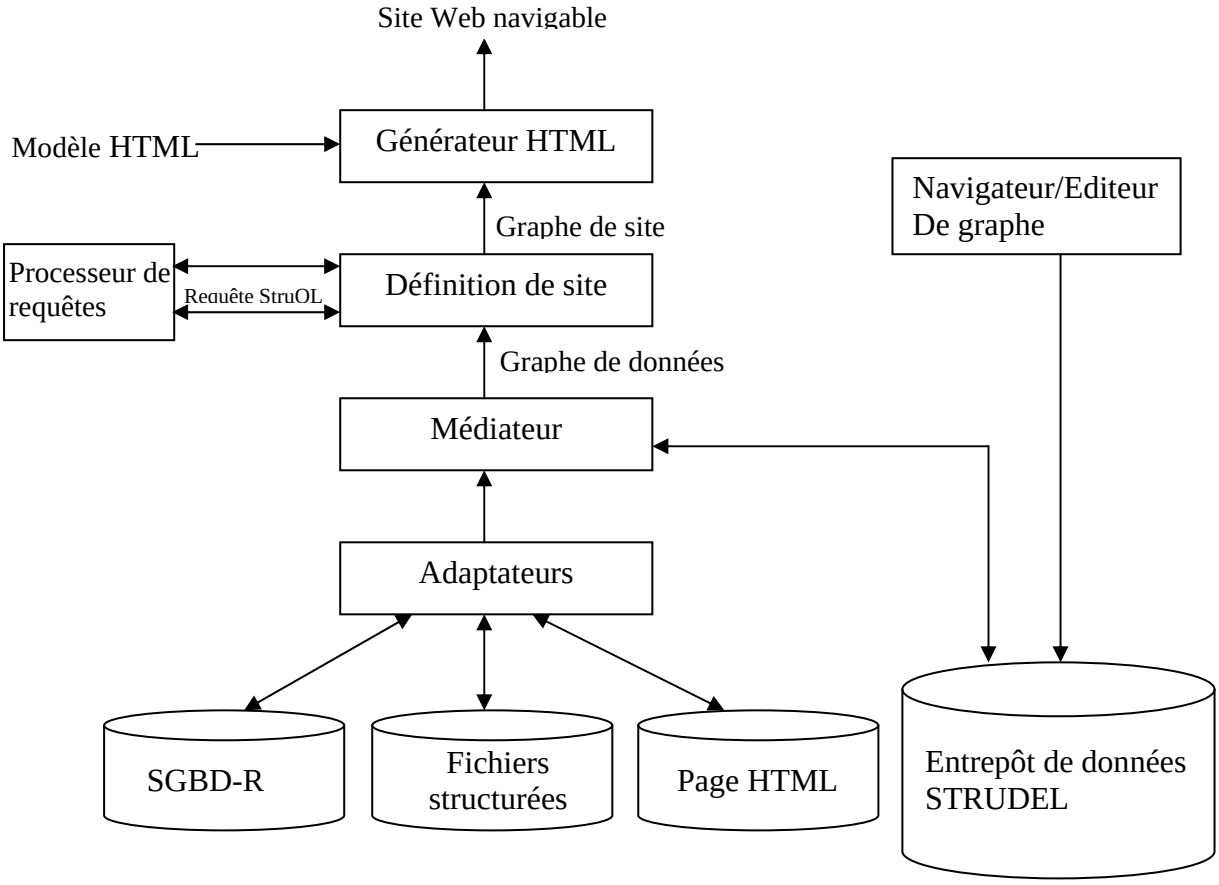


Figure A.2- Architecture de STRUDEL

A.1.3. YAT

De la même façon que STRUDEL, YAT intègre des sources de données dans un environnement Web. Il se base sur une représentation de graphes afin de représenter les données semi-structurées provenant des différentes sources. Le langage YATL est un langage déclaratif de conversion/intégration à base de règle ; ce n'est pas un langage de requête et il ne fournit pas de possibilités de requêtes comme des expressions de chemins généralisées. Cependant, il permet des primitives de

restructuration et des fonctions de Skolem pour manipuler des identifiants et créer des graphes complexes.

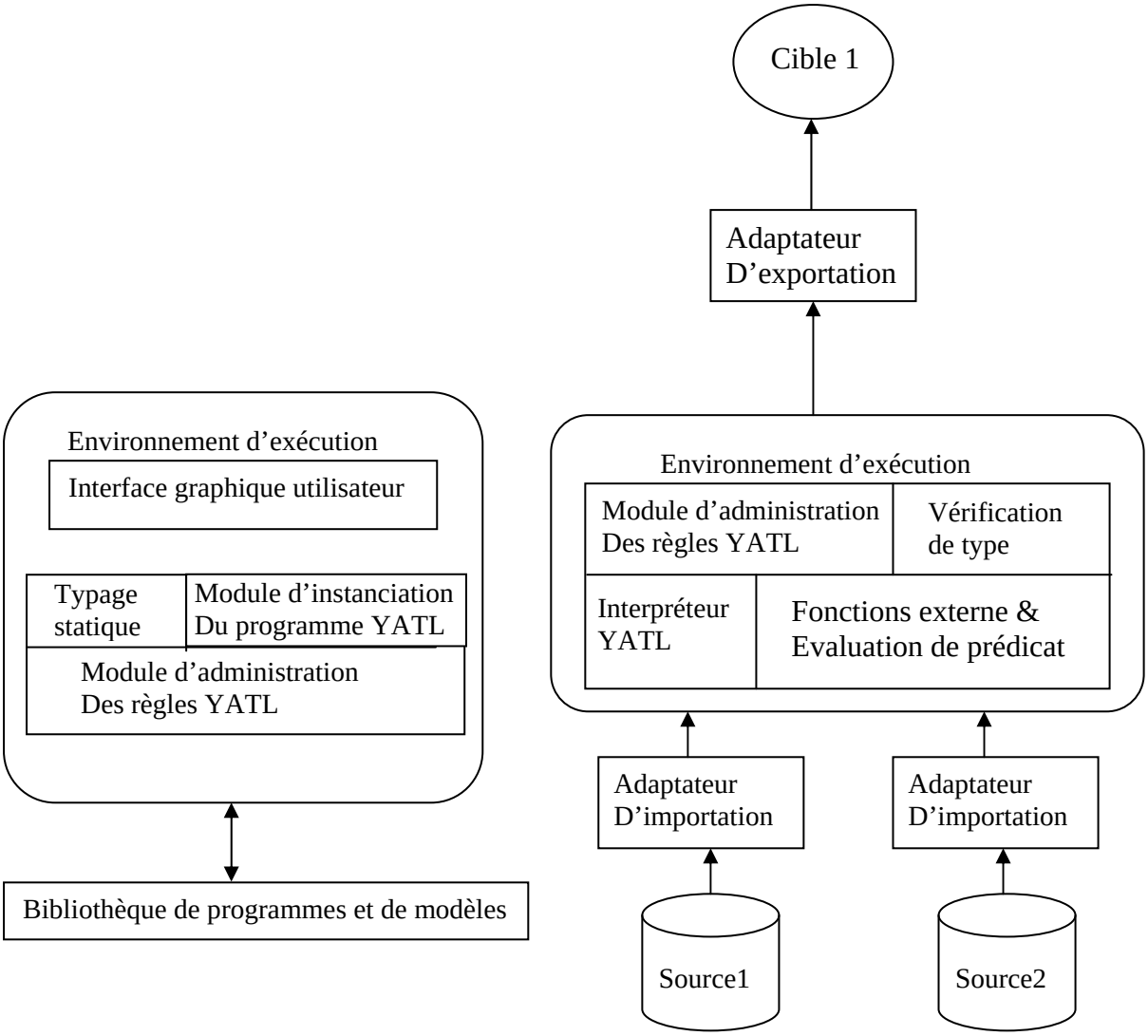


Figure A.3- Architecture du système YAT

La figure A.3 décrit l'architecture de YAT. Il y a trois parties principales : (1) l'environnement de spécification ; (2) l'environnement d'exécution, et (3) une bibliothèque de programme et format. Chacun des environnements repose sur le module de modèle YAT et d'administration de règles YATL. L'environnement de spécification offre : (i) un module permettant de vérifier ou déduire le type d'un programme, (ii) une interface graphique permettant à l'utilisateur de définir les traduction en utilisant (iii) module d'instanciation pour personnaliser les programmes existants si besoin est.
L'environnement d'exécution utilise des adaptateurs pour importer (resp. exporter) des données depuis (resp. vers) YAT et un interpréteur pour réaliser la traduction. L'interpréteur se base sur un module séparé pour traiter des fonctions externes et des prédicats. Si cela est requit, il vérifie les types à l'aide du vérificateur de type.

A.1.4. AGORA / LeSelect

AGORA [Manolescu et al. 2001] manipule des données semi-structurées. Pour cela, il s'appuie sur le médiateur LeSelect [Caravel 1998]. LeSelect est un médiateur dont le modèle de données est de type modèle relationnel étendu et dont le langage d'interrogation commun est de type SQL. La motivation de AGORA [Florescu et al. 2000] est de compléter les fonctionnalités de LeSelect en :

- Définissant un schéma virtuel, générique et relationnel décrivant le contenu de document XML.
- Traduisant des requêtes spécifiques à XML dans le schéma d'intégration relationnel.
- Etendant l'optimisation de requête afin de pouvoir gérer les cheminements
- Utilisant l'indexation textuelle pour améliorer les performances des recherches plein texte.

L'objectif d'AGORA est de supporter les requêtes et l'intégration de sources relationnelles et semi-structurées. Le schéma global de AGORA est une DTD XML. Les sources relationnelles et XML sont décrits comme vues sur ce schéma global. Un schéma relationnel générique est utilisé comme interface entre ce schéma et les sources. Pour chaque type de nœud XML (élément, attributs, textes, commentaire), une table relationnelle est associée dans ce schéma.

Dans cette façon, les requêtes XQuery sont transformées en requêtes relationnelles et les résultats sous formes de tuples sont ensuite transformés en XML.

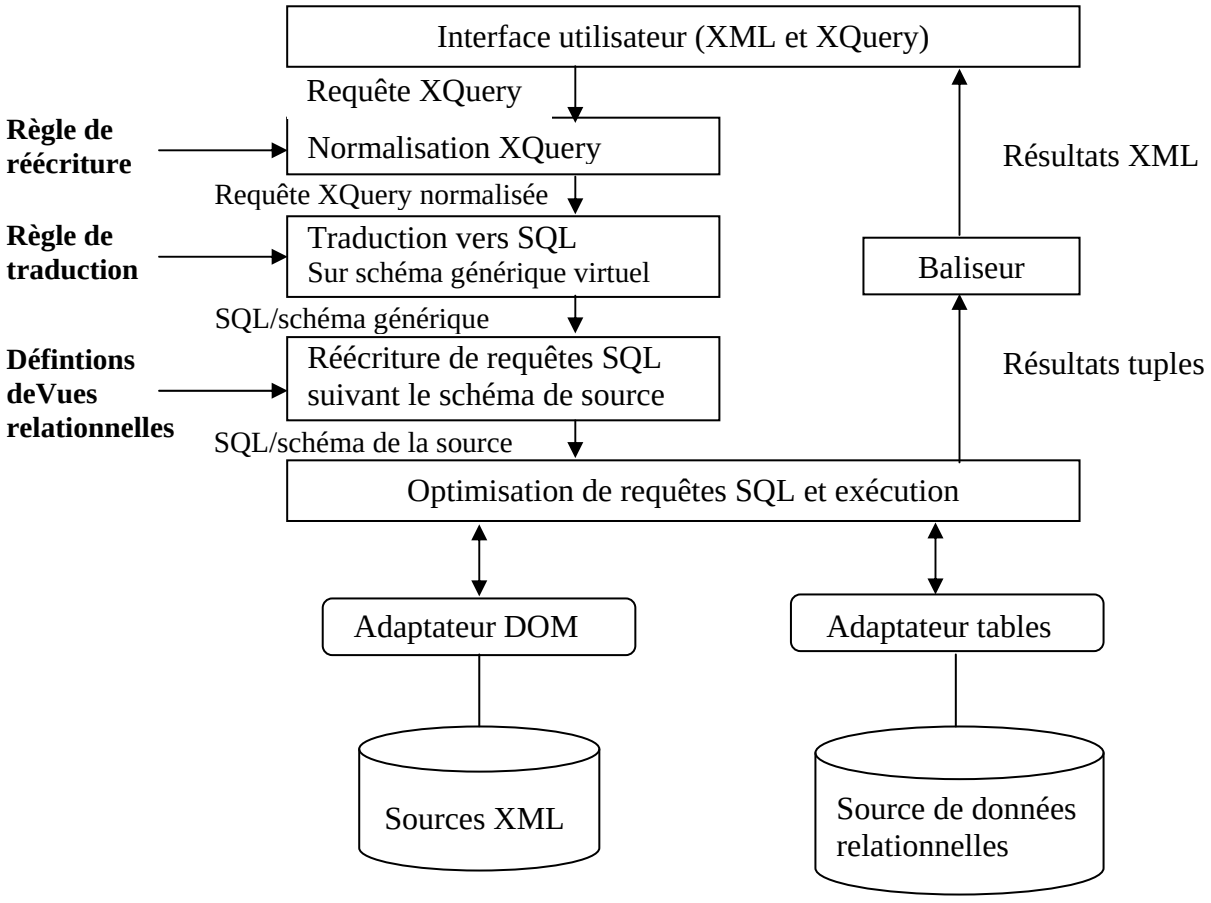


Figure A.4- Architecture du système AGORA

A.1.5. Logiciels commerciaux

Outre les projets de recherches que l'on a cités ci-dessus, de nombreuses entreprises ont conçu des systèmes de médiation de données semi-structurées. Suite à leur conception en milieu industriel, il est très difficile de trouver de la documentation quand à la conception et description des algorithmes utilisés dans ces logiciels. Nous ne détaillerons donc pas ces systèmes. On peut néanmoins citer les cas de XPeranto, NIMBLE et LiquidData qui bien que commerciaux, fournissent des spécifications techniques assez détaillées.

A.1.5.1. Xperanto

XPeranto [IBM 2002][Carey et al 2000a] (Xml Publishing of Enlities, Relationships, ANd Typed Object) est le nom de code de la future suite d'intégration de données d'IBM, Xeranto est compose d'un processeur de requête XQuery, d'un système de médiation permettant de récupérer des données hétérogènes de diverses sources de données (SGBD, documents de multiples formats), et enfin d'un composant à base de feuilles de style permettant de présenter les résultats sous une forme facilement intégrable dans des applications clientes. Comme AGORA, XPeranto [Carey et al 2000b] transforme un langage de requête orienté XML : XML-QL en SQL, pour cela, il passe par un langage intermédiaire de requête sur XML. Un composant XML Tagger permet ensuite de convertir la structure tabulaire des résultats SQL en document XML structuré.

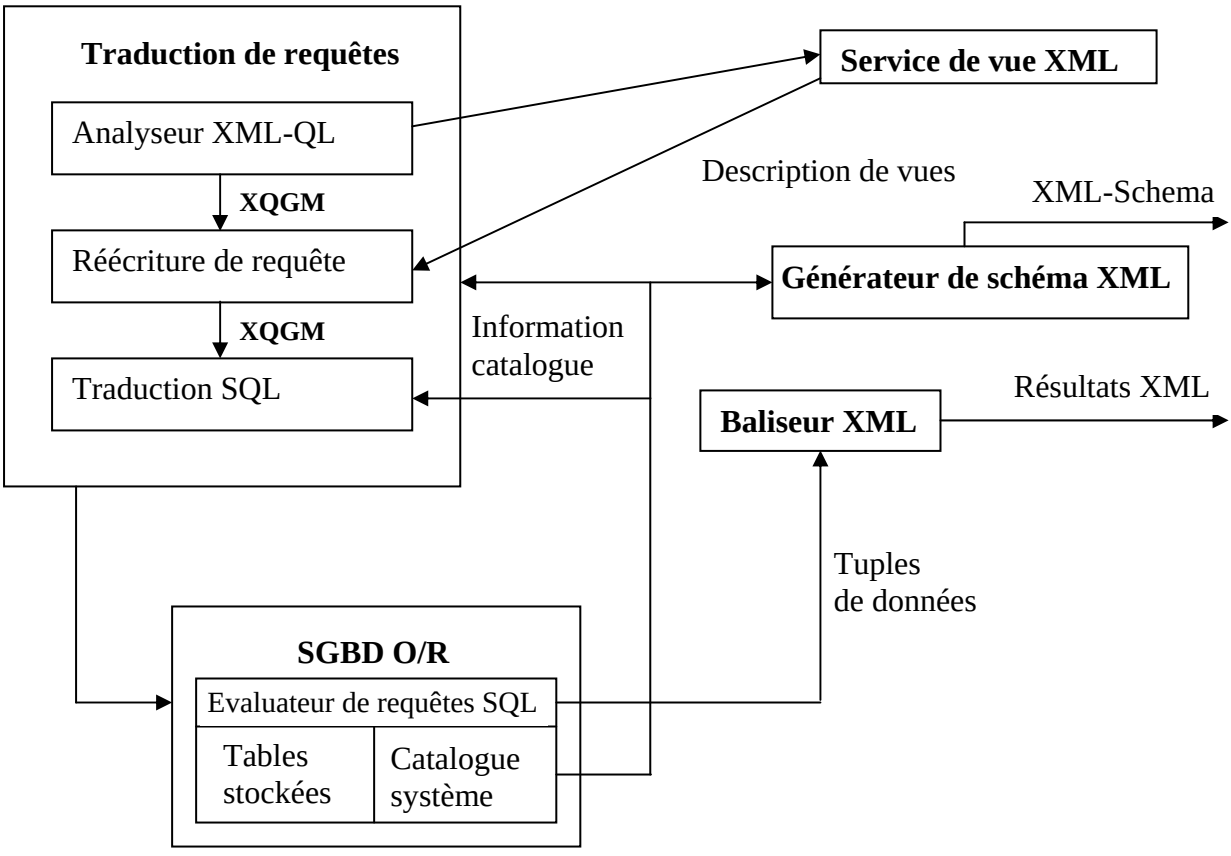


Figure A.5- Architecture de XPeranto

La figure A.5 décrit l'architecture d'XPeranto. L'architecture est composée de :
Un tradicteur de requêtes : qui traduit la requête XML-QL en langage XML-QL et génère une représentation XQGM (XML Query Graph Model). Le composant de réécriture de requête prend la présentation XQGM de la requête, résout les références sur les vue XML et construit une représentation sémantique équivalente XQGM. Le composant de traduction SQL traduit enfin le XQGM en instruction SQL.
Un service de vues XML : fournissant une interface de stockage et de chargement de données pour les définitions de vues XML-QL.
Un générateur de schéma XML : prenant les informations du catalogue des bases et produisant des informations pour les vues et résultats de requête XML
Un baliseur XML : convertissant un résultat tabulaire SQL en un document XML structuré.

A.1.5.2. NIMBLE

La suite NIMBLE [Draper et al 2001] est une plate forme d'intégration basée sur XML. NIMBLE utilise XML comme format commun pour représenter des données hétérogènes distribuées. La constatation faite par Nimble est que touts les modèles réalisés pour XML sont représentés sous forme de graphe ou d'arbre et que tous sont orientés pour manipuler du XML pur. L'idée est de proposer un modèle de données qui s'accommoderait d'XML, mais qui traiterait efficacement les types de données les plus utilisés (relationnels, hiérarchique). Le modèle de données de NIMBLE est adapté aux aspects semi-structurés de XML mais est légèrement plus structuré que le modèle décrit pour XML afin de gérer plus naturellement les données relationnelles et hiérarchiques. Le langage de requête utilisé est XML-QL.

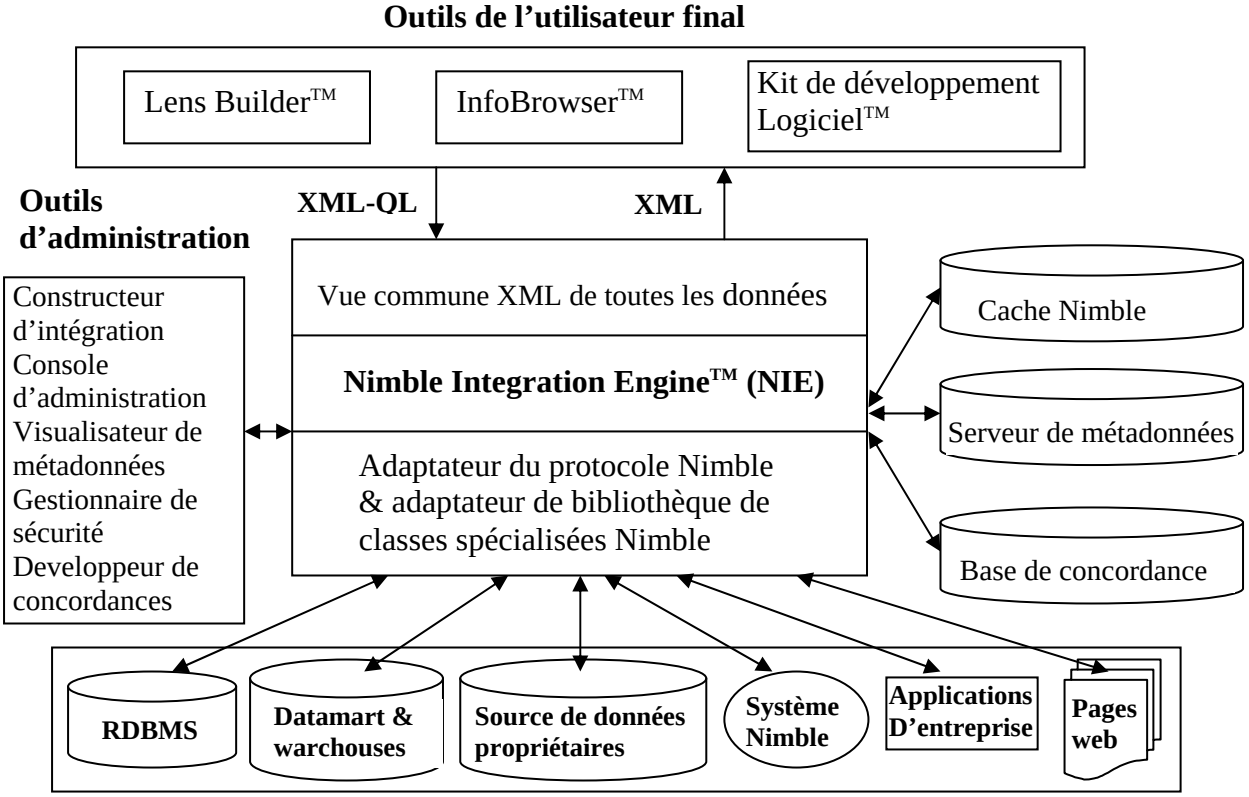


Figure A.6- Architecture de NIMBLE

La figure A.6 représente l'architecture de NIMBLE. Cette architecture est composée de :
Moteur d'intégration Nimble (Nimble Integration Engine) : il comporte un compilateur de langages de requêtes, une gestion de connexion aux clients et aux stockages phisiques. Le moteur d'intégration Nimble analyse et compile les requêtes en utilisant le serveur de métadonnées afin de déterminer quelle source de données est requise et si le cache Nimble doit être utilisé.
Cache Nimble : controlé par l'administrateur. Il permet de précalculer des vues XML sur une source et de stocker les documents XML résultants dans une base de données. Une requête future portant sur cette vue XML sera ainsi redirigée vers la version précalculée dans la base de données.
Base de concordance : permet de réconcilier les différences entre les diverses sources de données pour la représentation d'une même entité (ex. information sur la même personne représentée de deux façons différentes dans deux sources). Par un contrôle manuel de l'administrateur et par quelques règles prédéfinies, la base de concordance réconcilie ces différences en créant un index de clefs appropriés pointant vers les données originales.
Outils d'administration : pour gérer la base de concordance, le cache, le médiateur.
Outils de l'utilisateur final : pour formuler la requête de façon conviviale, développer des applications interagissant avec le moteur d'intégration de Nimble, etc.

A.1.5.3. Liquid Data

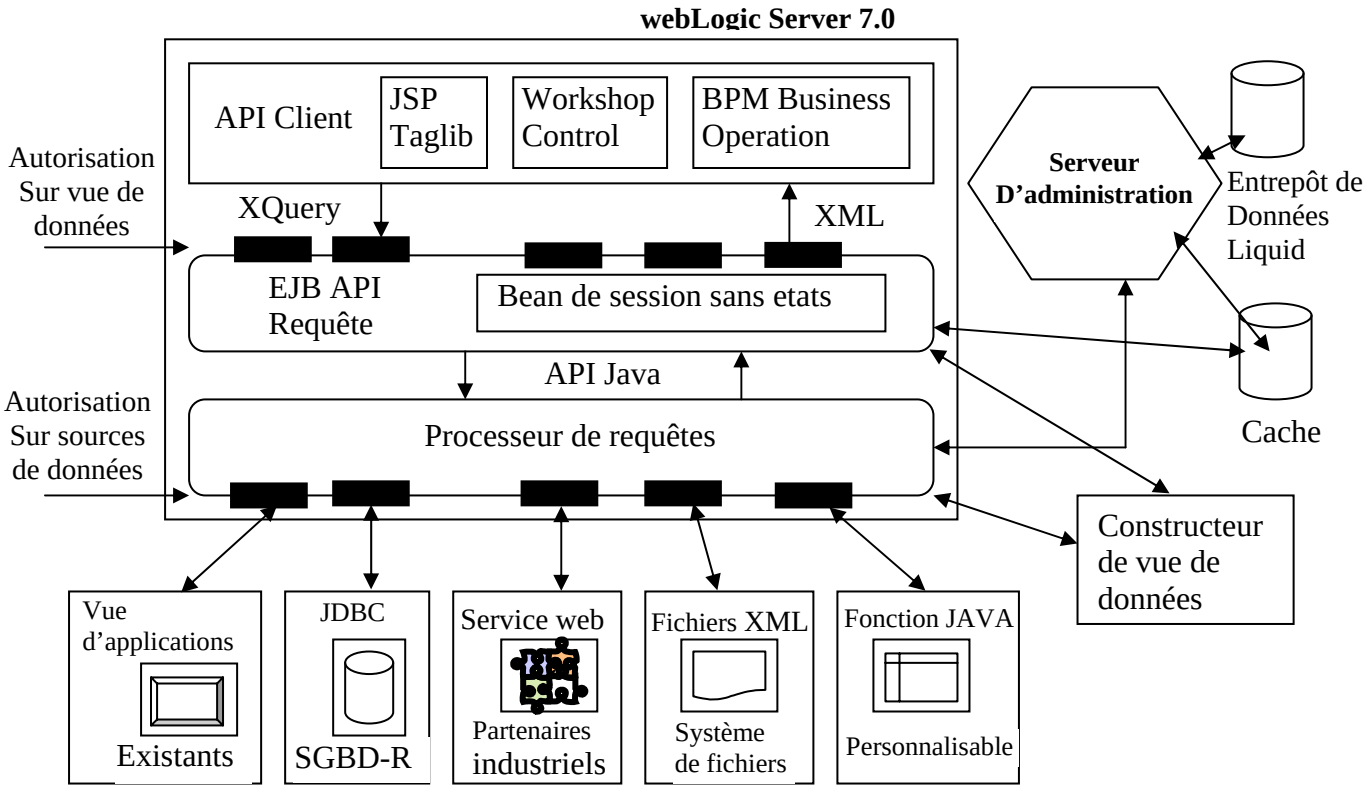


Figure A.7- Architecture de LiquidData

La figure A.7 décrit l'architecture de Liquid Data [BEA 2002]. Elle adopte l'architecture DARPA I3 avec un ensemble d'adaptateurs au dessus de sources de données hétérogènes, un modèle de données commun XML et un langage de requête commun XQuery. Les applications clientes accèdent au médiateur de LiquidData en envoyant des requêtes XQuery en un plan algébrique optimisé, qu'il transforme ensuite en un plan physique. Ce plan physique est ensuite exécuté sur les différentes sources via les adaptateurs, et les résultats sont remontés.

BIBLIOGRAPHIE

- **[ABI 97]:** S.Abiteboul, D.Quass, J.MeHugh, J.Widom et J.Wienner. The Lorel Query Language for Semi-Structured Data, April 1997.

- **[ABI 98]:** S. Abiteboul, J.Widoum, et T. Labiri. An Unified Approach for Querying Structured Data and XML, 1998 http://www.w3.org/TandS/QL/SL/SL98/pp/serge.html

- **[ABI 06]:** Serge Abiteboul, XML et données semi-structurées, Septembre 2006, http://www.rocq-inria.fr/~abitebou/DEA-III

- **[ALL 03]:** JM Alliot Qu'est-ce que le "Middleware", 13 mars 2003.

- **[AUS 02]:** R.Ausbrooks, S.Buswell, D.Carliste, S.Dalmas, S.Devitt, A, Diaz, R.Hunter, P.Ion, R.Miner, N.Poppelier, B.Smith,N.Soiffer. Mathematical Markup Language (MathML) Version 2.0, 2002. http://www.w3.org/TR/MathML2/

- **[BEA 02]:** BEA Liquid Data for WebLogic – Product Overview, Octobre 2002.

- **[BIR 00]:** P.Biron et A.Malhotra, XML Schema Part 2 : Datatypes, Octobre 2000.

- **[BRA 98]:** T.Bray, J.Paoli, et C.Sperberg-MacQueen. Extensible Markup Language (XML) 1.0 (W3C Recommendation), 1998.

- **[CAR 00a]:** M.J.Carey, D.Florescu, Z.G.Ives, Y.Lu, J.Shanmugasundaram, E.J. Shekita, et S.N.Subramanian. XPERANTO : Publishing Object-Relational Data as XML. WebDB(Informal Proceedings), 2000.

- **[CAR 00b]:** M.J.Carey, J.Kiernan, J.Shanmugasundaram, E.J.Shekita, et S.N.Subramanian. XPERANTO : Middleware for Publishing Object-Relational Data as XML Documents, In The VLDB journal, 2000.

- **[CHA 94]:** S.Chawathe, H.Garcia-Molina, J.Hammer, K.Ireland, Y.Papakonstantinou, J.Ulman, et J.Widom. The TSIMMIS Project-Integration of Heterogenous Information Sources. In 10th Anniversary Meeting of information Processing Society of Japan, Tokyo, Japan, 1994.

- **[CLA 01]:** J.Clark et S.Deach.
  Extensible Stylesheet Language (XSL), 2001.
- **[CLU 98]:** S.Cluet.
  Your Mediators need Data Conversion !
  In International Conference on Management of Data ACM SIGMOD, Seattle, 1998.

- **[DRA 01]:** D.Drapper, A.Y.Halevy, et D.S.Weld.
  The Nimble Integration Engine.
  In SIGMOD Conference, 2001.

- **[DRA 05]:** Florin DRAGAN, Georges GARDARIN,
  BENCHMARKING AN XML MEDIATOR,
  PRiSM Laboratory University of Versailles, France 2005

- **[FER 98]:** M.Fernandez, D.Florescu, J.Kang, A.Y.Levy, et D.Suciu Catching the Boat with Strudel: Experiences with a Web-Site Management System. In SIGMOD Conference, 1998.

- **[FLO 00]:** D.Florescu, I.Manolescu, D.Kossmann, et F.Xhumari. Agora : Living with XML and Relatinal. In Proceeding of the Conference on Very Large Data Base, Cairo, Egypt, février 2000.

- **[GAR 02]:** Georges Gardarin,
  XML : Des bases de données aux services Web, Dunod, 2002.

- **[GAR 06a]:** Geoarge Gardarin,
  Intégration de données et application via XML, 2006.

- **[GAR 06b]:** George Gardarin,
  Web services et mediation,
  PRISM Laboratory, UVSQ, Versailles France 2006.

- **[GEN 06]:** Genoveva Vergas Solar,
  Médiation de données : solutions et problèmes ouverts,
  CNRS, LSR-IMAGE, GRENOBLE 2006.

- **[HAL 00]:** A.Halvy. logic-based techniques in data integration. Logic based Artifical Intelligence, 2000.

- **[IBM 02]:** IBM.XTables,
  Bridging Relation Technology and XML, 2002.

- **[INC 97]:** Sun Microsystem Inc.
  JDBC: A java SQL API, 1997.

- **[JEA 02]:** Jean-Marie Chauvet
  Service Web avec SOAP, WSDL, UDDI, ebXML,
  Eyrolles 2002.

- |**MAN 01]:** L.Manolescu, D.Florescu, et D.Kossmann.
  Answering XML Queries over Heterogeneous Data Sources.
  In Proceedings of 27th International Conference on Very Large Data Bases (VLDB), Rome, Italie, September 2001.

- **[NAG 05]:** Meenakshi Nagarajan, Kunal Verma, Amit P. Sheth, John Miller,
  Semantic Interoperability of Web Services – Challenges and Experiences
  LSDIS Lab, Department Of Computer Science, University of Georgia, Athens GA, USA 2005.

- **[NGO 03] :** Tuyêt Trâm DANG NGOC,
  Fédération de données semi-structurées avec XML,
  Thèse de Doctorat, Soutenue juin 2003.

- **[PAP 95] :** Y.Papakonstantinou, H.Garcia-Molina, et J.Widom.
  Object Exchange across heterogeneous Information Sources. In int. Conference on data energineering, Taipei, 1995.

- **[PHI 00]:** Philips Jost,
  Analyse et synthèse de XML, 15 mars 2000.

- **[PHI 04]:** Philippe Michiels
  "XQuery Optimization"
  University of Antwerp, Belgium, 2004

- **[SUS 99]:** Susanne Busse, Ralf-Detlef Kutsche, UlfLeser, Herbert Weber,
  Federated Information System: Concepts, Terminology and architectures p2p,
  University Berlin, 1999

- **[THO 01]:** H.Thompson, D.Beech, M.Maloney, et N.Mendelsohn,
  XML Schema Part 1 :Structures, May 2001.

- **[VER 01]:** Hervé VERJUS
  Conception et construction de fédérations de progiciels,
  Thèse de doctorat présentée à l'université de SAVOI, le 24 septembre 2001.

- **[WIE 92]:** G.Wiederhold,
  Mediators in the Architecture of Future Information System. Computer, 1992

- **[WIK 06]:** Wikipédia,
  MathML, Article de l'encyclopédie libre, 8 juin 2006.
  http://fr.wikipedia.org/wiki/MathML

- **[W3C 06]:** W3C proposed recommendation 21 november 2006 ,
  XQuery 1.0: An XML Query Language
  http://www.w3.org/TR/2006/PR-XQuery-20061121/

GLOSSAIRE

A

Adaptateur XML (XML Adapter) : Module transformant une source de données non XML en XML.

Algèbre XQuery (XQuery Algebra) : Ensemble d'opérateurs de base permettant de représenter une requête XQuery par un plan d'opérateurs sur collections.

Analyse de données (Data analysis) : Ensemble de techniques permettant d'extraire des informations d'une base de données historisées par application de méthodes statistiques.

Annuaire de service (Service Registry) : Annuaire répertoriant et décrivant un ensemble de services.

Annuaire UDDI (UDDI Registry) : Annuaire de services Web accessible comme un service Web.

API (Application Program Interface) : Interface exposée par un programme et permettant de l'appeler depuis un autre programme.

API de communication (Communication API) : Interface pour les applications permettant de communiquer.

API de service (Container service API) : Interface pour les applications permettant d'invoquer des services souvent systèmes.

API HTTP (HTTP API) : API de serveur Web permettant d'invoquer des applications depuis HTTP.

Applet Java (Java applet) : Petit programme Java téléchargé et exécuté par un client Web.

Arbre DOM (DOM tree) : Arbre d'objets permettant de représenter un document XML ou HTML en mémoire.

Arbre DOM persistant (Persistent DOM tree) : Arbre DOM stocké de manière permanente en base de données.

ASP (Active Server Page) : Technologie Microsoft permettant la création de pages Web dynamique côté serveur par interprétation de templates HTML.

ASP (Application Service Provider) : Société louant des applications sur Internet accessibles par des connexions sécurisées; les applications sont de plus en plus présentées comme des services Web.

Attribut XML (XML Attribute) : Couple (nom="valeur") associé à une balise XML permettent de définir les caractéristiques d'un élément, jouant souvent un rôle descriptif par rapport à l'élément.

B

B2B (Business to Business) : Caractérise le type de commerce électronique conduit entre entreprises.

B2C (Business to Consumer) : Caractérise le type de commerce électronique conduit entre une entreprise et une personne privée.

Balise XML (XML label) : Aussi appelée tag ; mot entre chevron « < » et « > » permettant de délimiter un élément de données, la balise d'ouverture de l'élément est du type <nom_balise> alors que celle de fermeture est du type </nom_balise>.

Base de données fédérées (Federated database) : Base de données réparie composée d'un ensemble de bases de données hétérogènes.

Bean : Composant Java.

Bean entité (Entity Bean) : Composant Java avec état persistant.

Bean session (Session Bean) : Composant Java ayant au plus la durée de vie d'une session, sans état persistant.

BizTalk (BizTalk) : Organisme sponsorisé par Microsoft pour définir des schémas de messages XML dans de multiples domaine et les mettre à disposition sur Internet.

Bruit d'une recherche (Retrieval noise) : Ratio du nombre de résultats incorrects sur le nombre de résultats total de la recherche.

Business Process Reengineering (BPR) : Analyse et redéfinition des processus d'une entreprise menant à une définition des processus élémentaires, de leurs combinaisons, des messages échangés et plus généralement du système d'information.

C

COM (Common Object Model) : Architecture de Microsoft pour invoquer des objets distants depuis un poste client, s'appuyant sur un protocole binaire propriétaire.

Composant objet (Object Component) : Objet réutilisable fournissant une interface publique précise conforme à un standard permettant de l'assembler avec ses congénères pour développer des applications.

Communication asynchrone (Asynchronous Communication) : Type d'échange non bloquant dans lequel une application envoie un message à l'autre sans attendre la réponse, de sorte que les applications peuvent opérer indépendamment.

Communication synchrone (Synchronuous Communication) : Type d'échange bloquant dans lequel une application envoie un message à l'autre en attendant la réponse, de sorte que les applications opère en dépendance.

CORBA (Common Object Request Broker Architecture) : Standard de l'OMG permettant notamment l'invocation d'objets distants depuis un client, de manière transparente à l'utilisateur.

CSS (Cascading Style Sheets) : Feuilles de style en cascade basée sur l'association à une balise d'une présentation, permettant la mise en page de documents HTML et XML en fixant les attributs d'affichage (police, taille, couleur, etc.).

Cache de données (Data cache) : Mécanisme consistant à garder dans une zone tampon en mémoire (le cache) les données fréquemment accédées.

Capacité d'adaptateur (Adapter capabilities) : Fonctionnalités d'un adaptateur à pouvoir exécuter différents types de requêtes.

Chef d'orchestre (Orchestrator) : Module assurant l'exécution et la coordination d'activités, notamment dans un EAI.

Chiffrement XML (XML Encryption) : Technique de cryptage adaptée à XML.

Codage MIME (MIME encoding) : Codage des message selon les types des constituants sur HTTP.

Code virtuel Java (Bytecode) : Code Java intermédiaire exécutable.

Cohorte (Cohort) : Transaction longue composée de transactions courtes ou longues imbriquées.

Composant EJB (EJB component) : Composant Java exécutable côté serveur.

Composition de services (Services Composition) : Technique d'enchaînement de service Web pour composer de nouveaux services Web.

Connecteur XML (XML connector) : Module permettant de s'interfacer avec une application en XML, voisin d'adaptateur.

Container J2EE (J2EE Container) : Moteur d'exécution de composant sur un serveur d'application J2EE.

Contrat de composant (Component contract) : Interface à respecter entre un composant et le container avec J2EE.

Contrôle ActiveX (ActiveX control) : Composant COM sous Windows de Microsoft.

Coupe d'un cube (Cube slice) : Sélection d'un cube selon une dimension en OLAP.

Cube de données (Data cube) : Représentation tri-dimensionnelle d'un indicateur dans le domaine OLAP.

D

DataWebhouse (DataWebhouse) : Entrepôt de données en partie constitué de données issues du Web.

DCE (Distributed Computing Environment) : Architecture de systèmes distribués développée vers 1980 par l'Open Software Foundation (OSF), integrant des techniques fondamentales tel le RPC, les services de nommination, de temps synchronisé global, de fichiers distribués et de sécurité avancée.

DCOM (Distributed Common Object Model) : Extension de COM à l'invocation d'objets distribués sur un réseau.

Déclencheur (Trigger) : Règle définie au niveau du schéma d'une base de données, permettant d'invoquer une action composée d'une requête ou d'une procédure lorsqu'un événement apparaît sur les données.

Demandeur de service (Service Requester) : Application invoquant l'exécution de services, notamment de services Web.

Dépliage d'un cube (Cube drilldown) : Extension d'un cube en zoomant sur une dimension.

Description de service (Service Description) : Définition d'un service en termes d'interface, notamment avec WSDL pour les services Web ou IDL pour les services objet.

Dictionnaire de méta-données (Méta-data dictionnary) : Dictionnaire contenant des données décrivant de données.

Document bien formé (Well-formed Document) : Document XML respectant les règles de construction et la syntaxe définies dans le standard XML, notamment ouvrant et fermant correctement les balises et les imbriquant correctement sans chevauchement.

Document valide (Valid document) : Document bien formé conforme à une DTD ou un schéma XML.

DOM (Document Object Model) : Modèle objet de documents standardisé par le W3C, adapté à XML et HTML dynamique.

DRDA (Distributed Relational Database Architecture) : Architecture d'IBM permettant l'invocation et l'exécution de requêtes SQL distribuées.

DTD (Document Type Definition) : Définition de la grammaire d'un document XML en termes d'éléments et d'attributs.

E

EAI (Enterprise Application Integrator) : Outil permettant l'échange et l'intégration d'information entre différentes applications au sein d'une entreprise ou entre entreprise.

e-Business (e-Business) : Ensemble de procédés basés sur les échanges de données via Internet/Intranet et le Web pour conduire les processus de commerce du monde des affaires.

ebXML (ebXML) : Groupe de standardisation des technologies de l'e-Businness, patronné par Oasis et EDIFACT, visant notamment à redéfinir l'EDI en XML.

EDI (Electronic Data Interchange) : Echange de données électronique organisé selon des messages à plusieurs niveaux, avec en-têtes de trois caractères et des codages longueur – champ, standardisé dans les années 80.

EDIFACT (Electronic Data Interchange For Administration, Commerce and Transport) : Organisme dépendant de l'ONU définissant les standards de l'EDI.

Element XML (XML Element): Portion de document XML commençant par une balise d'ouverture (<balise) et se terminant par une balise XML (</balise>).

Entité XML (XML Entity) : Sorte de macros XML permettant de représenter par un nom d'entité des groupes d'éléments aussi bien pour les données (entités simples invoquée par &marco ;) que pour les DTD (entités paramètres invoquées par %macro ;).

Entrepôt de données (Data warehouse) : Base de données conservant des informations historisées organisées par sujet sur une mchine particulière**.**

Enveloppe SOAP (SOAP Enveloppe) : Elément racine des messages SOAP contenant un en-tête et un corps de message.

e-Procurment (e-Procurement) : Ensemble de procédés basés sur les échanges de données via Internet/Intranet et le Web pour conduire les processus d'approvisionnement dans l'entreprise.

Espace de noms XML (XML namespace) : Parfois appelé domaine de noms, ce concept permet de préfixer une balise par une chaîne identifiant sans ambiguïté sa définition (son

créateur) ; tous les éléments appartenant à un espace de noms sont préfixés par son identifiant suivi du signe deux-points (:) ; il est possible d'intégrer des balises provenant de plusieurs espaces de noms dans un même document XML.

ETL (ETL) : Outil d'extraction depuis une source, de transformation et de chargement de données dans un entrepôt.

Expression de chemin (Path expression) : Expression de balises successives permettant de sélectionner des nœuds dans un arbre XML par navigations élémentaires suivant les balises.

Expression de chemin généralisée (Generalized path expression) : Expression de chemins avec jokers et expressions régulières; XPath permet d'exprimer de telles expressions.

Extracteur de source (Source extractor) : Module permettant d'extraire des données depuis une source et de les publier selon un format convenu, par exemple XML.

F

Feuille de style (Style sheet) : Ensemble de commandes de mise en page associées à un contenu et précisant la présentation désirée.

Forêt XML (XML Forest) : Collection de documents XML, donc d'arbres d'éléments.

Fouille de données (Data mining) : Discipline consistant à extraire des connaissances à partir d'ensemble de données.

Fournisseur de service (Service Provider) : Site Web ou organisme mettant à disposition des services Web.

Fragment XML (XML Fragment) : Portion de documents XML pouvant être composée de un ou plusieurs éléments et pouvant posséder plusieurs racines.

G

GED (Gestion de Documents) : Ensemble des techniques permettant la gestion des documents dans l'entreprise, incluant notamment la gestion de bases de données documentaires.

Graphe OEM (Object Exchange Model Graph) : Graphe représentant un objet semi-structuré (document XML) sous forme d'agrégations et associations d'objets composants, les arcs étant étiqueté par les balises.

Guide de données (Dataguide) : Schéma faible d'une collection de documents extrait par analyse des chemins de balises et attributs.

H

HTML (Hyper Text Markup Language) : Le langage de balisage initial du Web, base sur des tags fixes et des contenus mixant données de fond et données de présentation, généralement interprété par un navigateur (Browser Internet Explorer ou Netscape).

HTTP (Hyper Text Transfer protocol) : Le protocole de transmission de pages Web d'Internet au-dessus de TCP/IP

I

IIOP (Internet Inter-ORB Protocol) : Protocol de l'OMG permettant l'invocation d'objets distants, sur lequel s'appuie l'architecture CORBA.

Index de chemin (Path index) : Index permettant de retrouver les éléments désigné par une expression de chemin à partir de valeurs des feuilles dans un document XML.

Indexation plein texte (Full text indexing) : Indexation par mots-clés et concepts d'un élément textuel.

Instruction XSL (XSL statement) : Commande XSL permettant de générer des données en sortie d'une règle.

Intégrateur de données (Data integrator) : Module capable d'intégrer plusieurs sources ou flux de données.

Intégration d'applications (Application intégration) : Forme de coopération entre systèmes d'information basée sur l'échange de données entre applications.

Intégration de données (Data Integration) : Forme de coopération entre systèmes d'information basée sur la fourniture de vues homogènes de sources de données hétérogènes aux applications clientes.

Intégration de schémas (Schema integration) : Processus consistant à rendre cohérent différents schémas et à composer un schéma unifié.

Intégration par modèle global (Global model integration) : Intégration de sources de données en les traduisant toutes dans un même modèle (souvent XML).

Intégration point à point (Point to point integration) : Intégration de sources deux à deux, en traduisant l'une dans le modèle de l'autre.

Intégration sémantique de données (Sémantique data integration) : Intégration de données en traduisant selon un même modèle, avec rapprochement ou différentiation sémantique.

Intégration syntaxique de données (Syntactic data integration) : Intégration de données en traduisant selon un même modèle, sans rapprochement ou différentiation sémantique.

Intelligence économique (Business intelligence) : Ensemble de techniques favorisant l'aide à la décision et l'automatisation du business, incluant fouille de données, entrepôts de données, OLAP, etc.

Interchange EDI (EDI Interchange) : Fichier de messages EDI échangé entre participants, par exemple donneur d'ordres et fournisseur.

J

J2EE (Java 2 Entreprise Edition) : Spécifications du langage Java et de l'architecture pour serveur d'applications associée.

JDBC (Java DataBase Connectivity) : API permettant à une application Java d'accéder de manière uniforme à toute base de données SQL, intégrée à J2EE.

JSP (Java Serveur Page) : Technologie permettant de créer des pages Web dynamique intégrant Java et HTML.

L

Lemmatisation d'un texte (Texte lemmatization) : Transformation d'un texte en une suite de mots-clés codés selon leurs formes préférées.

Liaison WSDL (WSDL Binding) : Association d'un client de service Web à l'interface WSDL de ce service.

Librairie de balises (TagLib) : Ensemble de balises XML prédéfinies et permettant d'invoquer des opérations sur des objets dans un JSP.

Lien XML étendu (Extended XML link) : Lien XML entre documents de type M-N, avec au moins N>1.

Lien XML simple (Simple XML link) : Lien XML entre documents de 1 vers 1, comme avec HTML.

Liste inverse (Inverted list) : Liste permettant à partir de la valeur d'un mot de retrouver les documents contenant ce mot.

M

Mapping de données (Data Mapping) : Règles mettant en correspondance deux schémas de données et permettant de transformer les instances.

Matérialisation de données (Data materialization) : Concrétisation de données généralement issues de bases dans un espace disque.

Matrice de fréquences de termes (Term frequency matrix) : Matrice indiquant pour chaque terme sa fréquence d'apparition dans chaque document d'une collection.

Médiateur (Mediator) : Composant logiciel capable d'exécuter des requêtes sur des données provenant de sources multiples.

Message SOAP (SOAP Message) : Message XML de racine enveloppe permettant d'invoquer un service Web.

Message WSDL (WSDL Message) : Message XML décrit en WSDL permettant d'invoquer une opération d'un service Web (appelé à intégrer l'élément <body> d'un message SOAP).

Méta-données (Métadata) : Données décrivant d'autres données.

Méthode de rappel (Callback method) : Opération d'une application pouvant être appelée par un module interne suite à un événement.

Middleware XML (XML Middleware) : Logiciel d'échange utilisant des protocoles XML, généralement entre les sources de données et les applications clientes.

MIME (Multipurpose Internet Mail Extensions) : Norme de codage des types de données texte, image, vidéo, son, etc., pour transmission en messages intégrés sur HTTP.

MOM (Message-Oriented Middleware) : Terme générique pour désigner les logiciels de type middleware permettant l'échange d'information par des messages synchrones ou asynchrones.

O

OAG (Open Applications Group) : Consortium industriel proposant des standards pour l'intégration d'applications e-Business.

OASIS (OASIS) : Organisme ouvert pour définir des schémas de messages XML dans de multiples domaine, les promouvoir et les mettre à disposition sur Internet.

Objet persistent (Persistent object) : Objet sauvegardé dans la base de données, ayant de ce fait une durée de vie supérieure à celled u processus ayant créé l'objet.

ODBC (Open Database Connectivity) : API standard permettant l'accès aux bases de données relationnelles via SQL, notamment implémentée au sein de Windows de Microsoft.

ODMG (Object Database Management Group) : Consortium de fournisseurs de BD objet ayant spécifié les interfaces d'un SGBD objet vers 1995, notamment OQL et ODL.

OQL (Object Query Language) : Langage d'accès unifié pour les bases de données objet, défini par l'ODMG, mais aujourd'hui peu implémenté.

OMG (Object Management Group) : Consortium de vendeurs de technologies objet proposant des standards pour les objets, notamment CORBA.

Orchestration d'activité (Activity Orchestration) : Pilotage d'activités composant des processus d'affaires à l'aide d'un workflow et de modèles de transactions.

ORB (Object Request Broker) : Logiciel au coeur de l'architecture CORBA transmettant les invocations d'objets et les réponses des clients aux serveurs d'objets, et vice-versa, implémentant généralement le protocole IOP.

OTP (Open Trading Protocol) : Standard B2C développé par l'IETF pour le commerce électronique, basé sur XML.

P

Parseur (Parser) : en français, analyseur voire parseur ; module analysant et décodant les balises d'un document XML afin de permettre à l'application utilisant cet analyseur de traiter les données des éléments.

Procédure stockée (Stored Procedure) : procédure associée à une base de données permettant d'effectuer une fonction de service sur la base, comportant généralement plusieurs requêtes SQL.

Passerelle CGI (CGI Gateway) : Interface permettant d'invoquer un programme et de lui passer des paramètres depuis une URL longue sur HTTP.

Pertinence d'une recherche (Retrieval recall) : Ratio du nombre de réponses correctes retrouvées parmi toutes celles relevantes suite à une requête de recherche.

PHP (PHP) : Technologie permettant d'intégrer des programmes écrits dans un langage interprété simple au sein de pages Web côté serveur.

Pliage d'un cube (Cube rollup) : Transformation d'un cube de données en réduisant l'unité selon une dimension dans le domaine de l'OLAP.

Pointeur XML (XPointer) : Adresse relative à l'intérieur d'un document XML souvent exprimée sous forme de XPath.

Port WSDL (WSDL Port) : Point d'accès à un groupe d'opérations d'un service Web.

Portail (Portal) : Point d'accès unique et uniforme sur le Web à un groupe d'applications ou de sources de données.

Portail de services (Web service portal) : Portail permettant d'accéder à un groupe d'applications se présentant sous forme de services Web.

Portail d'entreprise (Enterprise portal) : Portail permettant d'accéder à un groupe d'applications d'entreprise.

Portlet (Portlet) : Technologie Java basée sur les servler pour présenter et encapsuler une application destinée à un portail.

Précision d'une recherche (Retrieval precision) : Ratio du nombre de réponses correctes sur le nombre de documents retrouvés suite à une requête de recherche.

Processeur XSL (XSL processor) : Processeur de feuilles de styles écrites en XSL, capable de générer une présentation à partir d'une feuille de style et du contenu XML associé.

Processus d'affaires (Business Process) : Ensemble d'activités distribuées sur Internet ou intranet permettant de réaliser une transaction de commerce électronique entre partenaires.

Protocole HTTP (HTTP Protocol) : voir HTTP

Protocole SOAP (SOAP Protocol) : voir SOAP

R

RDA (Remote Data Access) : Protocole standard niveau application permettant d'envoyer des requêtes SQL à un serveur, de récupérer les réponses et de gérer les connexions.

RDF (Resource Description Framework) : Cadre XML pour définir des descripteurs de ressources (méta-données) par un nom de propriété et une valeur, par exemple auteur = "Hugo".

Rosettanet (Rosettanet) : Consortium industriel définissant des cadres de messages XML pour l'industrie et gérant un référentiel de formats sur Internet.

RPC (Remote Procedure Call) : Middleware orienté message permettant l'invocation par un simple appel d'une procédure distante depuis un client et le retour des paramètres de manière transparente; il permet de déporter au niveau du client une API serveur.

Racinisation d'un texte (Text steming) : Transformation d'un texte en une suite de mots-clés codés selon leurs racines.

Recherche documentaire (Document retrieval) : Recherche par liste, séquence ou disjonction de mots clés dans des collections de documents.

Réplication de données (Data replication) : Copie de données maintenues cohérentes dans plusieurs bases.

Requête distribuée (Distributed request) : Requête nécessitant l'accès à plusieurs bases sur des sites géographiquement dispersés.

Requête FLWR (FLWR query) : Expression de la forme "for…let…where…return…" permettant d'exprimer une recherche complexe en XQuery.

Requête XPath (XPath Query) : Expression XPath interprétée comme une recherche sur une collection de documents.

Requête XUpdate (XUpdate request) : Requête de mise à jour de nœuds de documents XML désignés par un XPath.

S

Saga (Saga) : Transaction longue composée d'une séquence de transactions atomiques avec compensations possible pour chacune.

SAX (Simple API for XML) : API événementielle pour XML, utilisée via un parser SAX ; celui-ci traite le document XML comme un flux et renvoie les éléments qui composent le document comme des événements au fur et à mesure de la lecture.

Schéma d'échange (Exchange schema) : Schéma de données échangé entre deux partenires.

Schéma d'intégration (Integration schema) : Schéma de données permettant d'intégrer plusieurs sources.

Schéma en étoile (Star schema) : Schéma composé d'un élément ou d'une table racine et d'éléments ou de tables dépendantes à un seul niveau.

Schéma en flocon (Flock schema) : Schéma composé d'un élément ou d'une table racine et d'éléments ou de tables dépendantes à plusieurs niveaux.

Schéma exporté (Export schema) : Schéma de données correspondant au format des données publiées par un adaptateur ou un connecteur.

Schéma global (Global schema) : Schéma unifié d'intégration des données dans une base de données fédérée.

Schéma importé (Import schema) : Schéma de données correspondant au format des données reçu d'un adaptateur ou d'un connecteur par un médiateur ou intégrateur.

Schéma local (Local schema) : Schéma de données correspondant au format d'une source de données.

Schéma XML (XML Schema) : Définition de la structure d'un document XML et des types des données élémentaires constituants conformément au standard de même nom.

SCM (Supply Chain Management) : Ensemble des technologies traitant de la chaîne d'approvisionnement, permettant le suivi de la fabrication des produits, de la conception à la livraison.

Scriplet Java (Java scriplet) : Programme Java intégré dansune page JSP.

Script client (Client script) : Programme en langage de commande transféré dans une page Web et interprété par un navigateur.

Script serveur (Server script) : Programme en langage de commande interprété côté serveur généralement invoqué par CGI ou par une page active.

Service Web (Web Service) : Groupe de functions accessible par un client Web en SOAP et décrit en WSDL.

Services d'annuaires (Directory Services) : Service permettant aux clients de localiser les services disponibles sur les serveurs, pouvant obéir à des standards tels LDAP ou UDDI.

Servlet Java (Java servlet) : Programme Java accessible par une URL longue comme un service côté serveur.

SGBD XML natif (Native XML DBMS) : SGBD dédié à XML stockant des documents XML sans les décomposer et permettant des accès rapide via index.

SGML (Standard Generalized Markup Language) : Langage de balisage normalisé en 1988, utilisé pour la gestion de documents, dont sont issus HTML et XML.

Signature XML (XML signature) : Elément XML de sécurité ajouter à un message afin de garantir que l'envoyeur du message est bien celui qu'il prétend être.

Similarité de documents (Document similarity) : Mesure permettant d'évaluer la ressemblance entre deux documents.

SMTP (Simple Mail transfer protocol) : Protocole de transmission des courriers électroniques sur Internet.

SOAP (Simple Object Access Protocol) : Protocole permettant de faire communiquer des applications réparties en XML au-dessus de HTTP.

SQL (Structured Query Language) : Langage de requête standard du modèle relationnel, inventé par E. Cod en 1970.

SSO (Single Sign On) : Clé et mot de passé unique permettant d'identifier un utilisateur pour un groupe d'applications.

Syndication de contenu (Content syndication) : Intégration de différents contenus généralement XML au sein d'une page Web unique ou d'un groupe de pages Web.

Système de médiation de données (Data mediation system) : Système permettant d'interroger des sources de données hétérogènes et de fournir des données intégrées en résultats.

T

TCP/IP (Transmission Control Protocol/Internet Protocol) : Le protocole standard pour l'Internet propose par tous les systèmes; IP est le niveau réseau et TCP le niveau transport dans le modèle en couche OSI.

Template HTML (HTML template) : Page HTML paramétrée par des programmes ou scripts exécutés côté serveur avant l'envoi.

Template XSL (XSL template) : Règles de production XSL codée en XML par un élément <template> avec condition en attribut et production en contenu.

Thésaurus (Thesaurus) : Dictionnaire de mots significatifs, souvent ciblé sur un domaine, avec pour chaque mot racine, mot préféré, synonymes, généralisation, etc.

Traduction de requêtes (Query translation) : Réécriture d'une requête exprimée dans un langage en une ou plusieurs requête dans un autre langage.

Traduction de table en attributs (Table attribute translation) : Traduction d'une table relationnelle en XML, chaque ligne générant un élément avec des attributs correspondant aux colonnes de la ligne.

Traduction de table en éléments (Table element translation) : Traduction d'une table relationnelle en XML, chaque ligne générant un élément avec des éléments imbriqués correspondant aux colonnes de la ligne.

Transaction simple (Simple transaction) : Transaction ACID (Atomique, Cohérente, Isolée et Durable).

Transformateur de données (Data mapper) : Processeur capable de transformer des données conforme à un schéma en données conforme à un autre schéma.

Transformateur XML (XML mapper) : Processeur capable de transformer du XML en XML, en s'appuyant généralement sur une feuille de style XSL.

Transformation de données (Data Transformation) : Modification de la structure voir de la sémantique d'un ensemble de données depuis un schéma source conformément à un schéma cible.

Transformation par règles (Rule-based mapping) : Transformation de données source en données cible par application de règles de production, par exemple XSL.

Transformation par requêtes (Query-based mapping) : Transformation de données source en données cible par application de requêtes permettant d'extraire les données cible, par exemple XQuery.

Transformation par schémas (Schema-based mapping) : Transformation de données conforme à un schéma en données conforme à un autre schéma en suivant des règles de correspondances.

Transformation relationnelle de XML (XML Relational Mapping) : Transformation de données XML en lignes de tables.

Transparence à la localisation (Location transparency) : Invisibilité de la localisation des données pour les utilisateurs.

Traqueur de messages (Message tracker) : Module capable d'enregistrer des messages et de retrouver un message particulier ou des messages sur critères.

Type complexe (Complexe type) : Type de données construit à partir de types simples ou complexes par application de constructeurs de collections.

Type simple (Simple type) : Type de données élémentaire nativement intégré par exemple en XML, comme entier, réel, chaîne de caractères.

U

UDDI (Universal Description, Discovery and Integration) : Annuaire des services Web permettant de répertorier les definitions administratives, fonctionnelles et techniques des services disponibles.

UML (Unified Modeling Language) : Langage de modélisation objet composé d'un ensemble de diagrammes permettant de représenter un système d'information d'un point de vue statique (classe, association, etc.) que dynamique (cas d'usage, diagrammes d'états, etc.).

URI (Universal resource Identifier) : Adresse générique normalisée permettant d'identifier une ressource Internet, composée d'une chaîne de caractères obéissant à différente forme syntaxique, la plus courante étant <type_connexion> :// <serveur> / <ressource> /... Une URL est un cas particulier d'URI.

URL (Universal resource Locator) : Adresse logique normalisée d'une ressource Internet accessible via HTTP, conçue au départ pour lier des pages Web.

URL longue (Long URL) : URL suivie d'un nom de programme et d'une chaîne de paramètres (du type nom=valeur) permettant d'invoquer un programme côté serveur**.**

V

Validation deux phases (Two-Phase Commit) : Protocole et mécanisme de journalisation sous-jacent permettant de garantir l'atomicité de transactions distribuées, c'est-à-dire l'échec ou le succès sur tous les sites.

Virtualisation de données (Data virtualization) : Procédé permettant de définir des vues non matérialisées de données intégrées déduites de sources hétérogènes et supportant les requêtes sur ces vues.

Vue d'un cube (Cube view) : Projections d'un cube OLAP selon une ou deux dimensions avec agrégation des valeurs projetées.

Vue intégrée (Integrated view) : Vue unifiée de plusieurs sources de données hétérogènes.

W

W3C (World Wide Web Consortium) : Organisme de proposition et de normalisation des technologies, protocoles et langages du Web ; à son actif, la standardisation de XML, XPath, DOM, XSL, XQuery, HTML, HTTP, etc.

Workflow (Workflow) : Diagramme de flux proche d'un réseau de Petri permettant de décrire des enchaînements d'événements et d'actions séquentiels ou parallèles, et par extension logiciel gérant les enchaînements et synchronisation.

X

X/Open (X-Open) : Groupe ouvert et indépendant des constructeurs intégrant des standards varies, souvent de type middleware au sein d'un environnement système intégré, appelé Common Applications Environment (CAE).

XHTML (XHTML) : Langage de la famille XML similaire à HTML, proposé par le W3C, résultant d'un nettoyage de HTML afin de suivre les spécifications du W3C.

Xlink (XML Linking Language) : Langage basé sur XML permettant de définir des liens simples (1-1) ou complexes (M-N) entre documents.

XML (XML) : Méta-language développé par le W3C permettant de définir des langages de marquage de documents ou de messages, au centre d'un ensemble de standards dédiés à la communication dans les systèmes d'information.

XML Schema (XML Schema) : Langage de définition de schéma des documents XML proposé par le W3C.

XPath (XML Path Language) : Langage permettant d'adresser des éléments, attributs, valeurs, etc., par navigation à l'intérieur d'un document XML, standardisé par le W3C.

XPointer (XPointer) : Langage basé sur XML permettant de complémenter des liens XLink pour adresser des éléments à l'intérieur de documents.

XQuery (XQuery) : Langage de requêtes pour interroger des collections de documents XML proposé par le W3C.

XSL (eXtensible Stylesheet Langage) : Langage de feuille de style extensible basé sur des règles de production.

XSLFO (XSL Formating Object) : Langage inclus dans XSL permettant de transformer un document XML en une séquence de blocs formatés pour l'impression.

XSLT (XSL Transformation) : Langage inclus dans XSL permettant de transformer un document XML en un nouveau document XML ayant une structure éventuellement différente.

XUpdate (XUpdate) : Langage de requêtes pour interroger des collections de documents XML proposé par le groupe dbXML.