Analyse OLAP d'un entrepôt de documents XML

Fatma Abdelhedi, Landry Ntsama, Gilles Zurfluh

IRIT-SIG, Université de Toulouse 1 Capitole
2 rue du Doyen Gabriel Marti, 31042 Toulouse, France
prenom.nom@irit.fr

RESUME. Les systèmes OLAP basés sur des entrepôts de données sont aujourd'hui bien intégrés dans les organisations, ils facilitent le traitement et l'analyse de l'information pour la prise de décision. Le développement du Web a conduit à l'accroissement du volume de données traité, ainsi qu'à la diversification des sources de l'information. Ce problème de diversification a été en partie résolu grâce au langage XML. Celui-ci permet en effet le traitement et l'échange de données complexes et hétérogènes. Seulement c'est un format qui s'adapte mal aux systèmes OLAP et d'entrepôts classiques. De plus il n'existe à ce jour aucun standard permettant de répondre à cette problématique. Aussi nous avons développé un modèle multidimensionnel qui utilise le formalisme orienté objet UML pour décrire un entrepôt de documents XML orientés-document. Le schéma de cet entrepôt (appelé StarCD) représente la structure des documents à analyser, telle qu'elle est connue par le décideur. Et dans cet article nous présentons un nouveau langage d'analyse OLAP destiné aux décideurs, qui permet d'exprimer des requêtes complexes sur un entrepôt de documents XML décrit par un StarCD.

ABSTRACT. OLAP systems based on data warehouses are nowadays well integrated into organizations and they ease the process of information analysis for decision-making. The Web expansion led to increase the volume of the data handled, as well as the sources diversification. This problem was partially solved thanks to the XML format, which indeed allows processing and exchanging complex and heterogeneous data. Only, this format does not fit for classic OLAP systems and warehouse, furthermore there is not an existing standard allowing solving this issue. So we developed a multidimensional model which uses the object oriented model UML to describe a XML Document Warehouse (XDW). The warehouse schema (named StarCD) represents the documents structure in the source, such as it is known by the decision-maker. And we present in this article a new OLAP analysis language intended for decision-makers, which allows them to express complex query on a XDW described by its StarCD.

MOTS-CLES : OLAP, Modélisation multidimensionnelle, XQuery, Entrepôt de documents XML.

KEYWORDS: OLAP Systems, XML Warehouse, XQuery, Multidimensional modeling

1. Introduction

Depuis une vingtaine d'années, les entrepôts de données et les systèmes OLAP associés ont été développés afin de répondre aux exigences de la prise de décision. L'évolution rapide des nouvelles technologies, notamment du Web, a fait naître de nouvelles problématiques liées à l'exploitation de données complexes et hétérogènes. Le format XML *(W3C, 2013a)* permet le traitement et l'échange de ce type de données à travers le Web. Aussi son exploitation dans les systèmes décisionnels nécessite la définition de nouveaux modèles *(Pérez et al, 2008)*.

L'intégration des documents XML dans les entrepôts a fait l'objet de nombreux travaux, dont la majeure partie concerne les documents XML « orienté-données » *(Pérez et al, 2008; Pokorny, 2001 ; Golfarelli et al, 2001)*. D'autres travaux ont traité de l'intégration des documents « orientés-document » *(Nassis et al, 2005 ; Tseng et Chou, 2006)*, en proposant de nouveaux modèles multidimensionnels ((par exemple le modèle en galaxie), et en étendant les langages d'analyse existant (SQL, MDX ou XQuery) pour l'analyse OLAP de ces entrepôts. Certains de ces modèles proposent de déstructurer le document afin de le représenter dans le schéma multidimensionnel, rendant ainsi la structure méconnaissable pour l'utilisateur. De plus, le décideur étant un non informaticien, la manipulation d'un langage comme XQuery *(W3C, 2013d)* dans le contexte d'un entrepôt XML peut s'avérer délicate.

Notre solution consiste à créer un entrepôt de documents XML (orientés-document) à partir d'une source contenant une collection de documents thématique (dont les structures sont unifiées). Pour l'élaboration du schéma multidimensionnel, nous avons étendu le modèle en étoile classique *(Golfarelli et al, 1998)*, les faits, dimensions et mesures sont extraits des XML-Schémas (XSchémas) *(W3C, 2013c)* décrivant les documents de la source. La structure hiérarchique des documents est donc préservée et facilite la compréhension du schéma multidimensionnel par les décideurs. Mais la contribution principale de cet article est la définition d'un langage d'analyse OLAP qui permet à des décideurs d'analyser un entrepôt de documents XML. C'est un langage dont la syntaxe relativement simple repose sur les principes développés dans les langages de requêtes pour objets complexes *(Barry et Cattell, 1997)*.

L'article est organisé comme suit. La section 2 présente un état de l'art relatif à nos travaux présentés. La section 3 présente une définition formelle du modèle multidimensionnel. La section 4 présente le langage d'analyse pour les décideurs. La section 5 présente le processus de traduction et la section 6 conclut cet article en soulevant des points de perspectives.

2. Etat de l'art

Dans *Ravat et al. (2007)*, les auteurs proposent un nouveau modèle multidimensionnel pour l'analyse OLAP des documents XML. Ce modèle dit « en galaxie » permet d'élaborer un schéma d'entrepôt sous la forme d'un graphe de

dimensions. Les dimensions d'une galaxie sont liées entre elles par un ou plusieurs nœuds exprimant la compatibilité entre les dimensions. C'est au moment de l'interrogation de l'entrepôt que le fait est désigné parmi les dimensions. Ainsi une dimension dans un schéma en galaxie représente à la fois une dimension et un sujet d'analyse. L'auteur étend l'algèbre multidimensionnelle classique qu'il adapte à son modèle en y intégrant de nouveaux opérateurs notamment pour l'analyse de données textuelles. Dans ces travaux, le modèle de données proposé pour l'entrepôt ne préserve pas la structure initiale des documents. Les constituants (éléments) des documents sont déstructurés (indépendants tout en restant liés) dans le schéma de l'entrepôt et l'on peut raisonnablement penser que ceci peut constituer un obstacle dans l'expression des requêtes par les décideurs.

Dans *Nassis et al. (2005)*, les auteurs proposent de décrire un entrepôt de documents XML en utilisant le modèle des diagrammes de classes d'UML. Dans le schéma de l'entrepôt, les documents sont représentés par un fait lié à des dimensions virtuelles. Les auteurs mettent l'accent sur l'aspect modélisation, sans définir d'approche concrète pour l'analyse OLAP via leur modèle. Ils présentent brièvement un algorithme de requête basé sur XQuery pour analyser un entrepôt. Le modèle de données proposé dans l'article permet de décrire un entrepôt de documents, en termes de faits et dimensions, grâce à des classes d'objets distinctes liées entre elles, et organisées suivant les besoins de l'utilisateur. Ainsi la structure des documents, telle quelle apparaît dans la source, n'est pas conservée dans l'entrepôt. Or cette structure est connue des décideurs qui souhaitent analyser des documents.

Et *Park et al. (2005)* proposent un modèle de données identique au précédent pour modéliser un entrepôt. Ils proposent d'étendre le langage MDX qui est à la base un langage de requête OLAP pour les entrepôts de données classiques. Ce langage étendu (XML-MDX) intègre une série d'opérateurs pour la manipulation d'un cube XML en permettant notamment l'analyse de contenus textuels. Le langage XQuery est utilisé lors la création du cube XML pour le calcul des mesures et la définition des dimensions. Tout comme dans l'approche présentée par *Nassis et al. (2005)*, la structure des documents à analyser ne se retrouve pas dans le schéma de l'entrepôt. De plus, le langage défini pour l'analyse OLAP s'avère complexe car il nécessite que le décideur ait des connaissances informatiques en manipulation de fichier XML (XQuery) ainsi qu'en analyse OLAP (MDX). Or celui-ci est par essence un non-informaticien.

Les approches présentées ci-dessus ne répondent que partiellement à notre problématique. Dans notre approche, nous supposons que les décideurs connaissent la structure des documents qu'ils désignent dans la source pour être analysés. Ils retrouveront cette structure dans le schéma de l'entrepôt ; et c'est cette structure qui leur permettra d'exprimer leurs requêtes d'analyse.

3. Définition du modèle multidimensionnel

3.1 Définition formelle

Le modèle que nous présentons dans cette section permet de décrire un entrepôt de documents XML sous la forme d'un schéma en étoile nommé StarCD. Ce modèle repose sur le formalisme UML *(OMG, 2013)* pour représenter le fait, ses dimensions et ses mesures. Il est élaboré à partir de l'analyse d'une source de documents XML décrits par un même schéma XML (XSchéma) et sert de support pour l'élaboration des requêtes d'analyse par les décideurs.

Un StarCD est défini à partir du XSchéma qui décrit les documents XML sous forme d'arbre :

StarCD = (F, D) où :

- F correspond au fait, c'est-à-dire à l'élément racine du XSchéma définissant la classe de documents à analyser.
- D = {D_1, …, D_n} est un ensemble de dimensions associé au fait.

Le fait est caractérisé par un ensemble de mesures, de types numérique ou textuel. Dans le StarCD, il représente l'ensemble des documents de la source à analyser. Les mesures sont décrites sous forme de classes liées au fait par des liens d'agrégation ; on retrouve ainsi la structure hiérarchique qui est présente dans le XSchéma de la source. Le fait est défini comme suit :

F = (M, Agg) où :

- M = {M_1, …, M_p} est un ensemble de mesures.
- Agg = {ag_1, …, ag_p} l'ensemble des fonctions d'agrégation associées aux mesures, avec *ag* dans {*SUM, COUNT, AVG, MAX, MIN*}.

Les dimensions sont quant à elles caractérisées par des ***paramètres*** organisés en hiérarchies. Ces hiérarchies précisent des niveaux de granularité allant du paramètre de plus haut niveau (All) au paramètre de plus bas niveau, celui-ci correspond à la classe liée au fait. Les dimensions sont définies comme suit :

D = {P, H} où :

- P = {p_1, …, p_s} est l'ensemble des paramètres de la dimension D
- H = {h_1, …, h_s} est l'ensemble des hiérarchies dans lesquelles sont organisés les paramètres, avec h défini comme suit :
    - h = {p_1, p_2, …, *All*} où p1 est le paramètre lié au fait (paramètre de plus bas niveau, et All étant le paramètre de plus haut niveau sur la hiérarchie.

Dans le StarCD, les dimensions sont décrites sous la forme d'un ensemble de classes ; chaque classe représente un paramètre qui permet de partitionner

l'ensemble des instances du fait. Chaque dimension est une arborescence de classes dont la racine est liée au fait par une relation d'association «By». C'est une relation d'association UML étendu pour indiquer l'axe choisi pour l'analyse. Les attributs faibles des dimensions correspondent aux attributs des classes.

Avec ce modèle nous introduisons le concept de ***hiérarchie inversée***. Dans ce type de hiérarchie, le paramètre de plus haute granularité est relié au fait pour respecter le XSchéma de la source. Le paramètre « All » considéré comme paramètre de plus haut niveau dans une hiérarchie « normale » n'est plus pris en compte ici.

3.2 Etude de cas

Pour illustrer notre démarche, nous disposons d'une collection d'articles scientifiques de même thématique que nous souhaitons analyser, et à partir de laquelle nous allons construire un entrepôt de documents. La source est décrite par un XSchéma (voir la Figure 1) représentant l'ensemble des documents, et la Figure 2 représente le StarCD qui en est extrait, contenant le fait, les dimensions et les mesures que nous avons sélectionnées pour cette analyse.

```
<?xml version="1.0" encoding="UTF-8"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema">
<xs:element name = "Paper">
  <xs:complexType>
    <xs:sequence>
        <!--Types simples -->
        <xs:element name="Title" type="xs:string"/>

        <!--Types complexes -->
        <xs:element name = "CoAuthors">
        <xs:complexType>
            <xs:sequence>
                <xs:element name="Author" maxOccurs="unbounded">
                <xs:complexType>
                    <xs:sequence>
                        <xs:element name = "Name" type="xs:string"/>
                        <xs:element name = "FirstName" type="xs:string"/>
                        <xs:element name = "Affiliation" type="xs:string" minOccurs="0"/>
                        <xs:element name = "Mail" type="xs:string" minOccurs="0"/>
                    </xs:sequence>
                    <!--Attributs auteur -->
                    <xs:attribute name = "H-Index" type="xs:integer" use ="required"/>
                  </xs:complexType>
              </xs:element>
              <xs:element name = "AffiliationGroup" type="xs:string" minOccurs="0"/>
          </xs:sequence>
      </xs:complexType>
    </xs:element>
    <xs:element name="Abstract" type="xs:string" minOccurs="0"/>
    <xs:element name="Résumé" type="xs:string" minOccurs="0"/>
    < !--    -->
```

Figure 1. XSchéma partiel de la classe de document Paper

Le fait et les mesures correspondent à l'arborescence de la racine Paper et sont situés dans la partie basse du StarCD. Les dimensions quant à elles figurent dans la partie haute. Comme pour les mesures, la structure hiérarchique des documents sources se retrouve dans la forêt des dimensions. Ceci contraint un sens particulier des liens hiérarchiques entre paramètres (***hiérarchie inversée***). Ainsi dans la Figure 1, la classe CoAuthors est directement liée au fait Paper parce que CoAuthors est un élément de premier niveau dans le XSchéma décrivant les documents sources ; les paramètres de cette dimension sont donc inversés par rapport au sens classique. De plus, d'après *Bordawekar et Lang (2005)* tout élément XML peut être analysé autant en tant que dimension que mesure. Ceci est dû à la classification entre les dimensions et les mesures qui n'est pas rigide, comme c'est le cas pour les données plates. Nous retrouvons cette propriété dans notre modèle, où chaque élément identifié comme dimension peut aussi être identifié comme mesure (par exemple la classe KeyWord dans la Figure 2). L'élément Date est néanmoins utilisé exclusivement comme dimension.

4 Langage d'analyse OLAP

4.1 Syntaxe et sémantique

Le langage de requêtes XQuery *(W3C, 2013d)* a été développé et standardisé par le W3C et il permet d'interroger des documents XML. Toutefois sa syntaxe est complexe et l'expression des requêtes parfois difficile. En effet, il nécessite une bonne connaissance d'une part du langage de sélection XPath *(W3C, 2013b)* permettant de naviguer dans la structure d'un document et d'autre part de la structure des fichiers XML. L'exemple 1 de requête suivant démontre la complexité du langage :

Exemple 1:

```
for $a in //DWordGroup/WordGroup,
    $b in distinct-values($a/KeyWord),
    $c in //DPubliDate/PubliDate,
    $d in $c/PubliMonth/PubliYear/@year
    let $doc := //Paper[WordGroup/@ref = $a/@id and
                        PubliDate/@ref = $c/@id]
return
   if (exists($doc))
   then <group>
            <KeyWord>{$b}</KeyWord>
            <PubliYear>{data($d)}</PubliYear>
            <val>{count($doc)}</val>   </group>
```

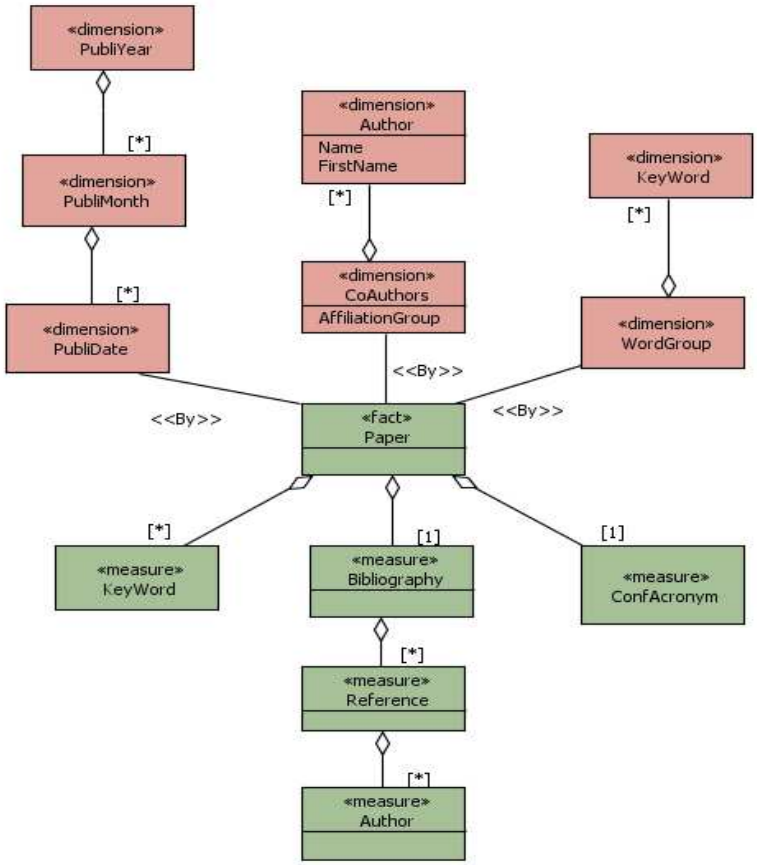

Figure 2. Diagramme en étoile StarCD

Aussi, le langage d'analyse présenté dans cette section permettra à des décideurs (non informaticiens) d'exprimer des requêtes complexes. Il s'inspire du langage OQL *(Barry et Cattell, 1997)* et permet de manipuler les objets représentés dans le schéma multidimensionnel StarCD. Sa structure facilite la navigation dans l'arborescence d'un fichier XML et sa syntaxe est de la forme suivante:

Analyse <*mesure*>

From <*variables désignant les classes de l'entrepôt*>

By <*dimensions*>

La clause Analyse contient les mesures ainsi que les fonctions d'agrégation qui leur sont appliquées. La clause From spécifie l'ensemble des éléments qui, au sein

du StarCD, sont utilisés soit comme mesure soit comme dimension ; chaque élément est associé à une variable de désignation (« alias ») qui permet d'éviter les ambiguïtés lors du parcours des hiérarchies. La clause By désigne les dimensions à utiliser comme axe d'analyse. La structure de la requête est définie suivant la grammaire BNF suivante :

<Query_spec> ::= <Analyse_clause><From_clause><By_clause>;

<Analyse_clause> ::= Analyse <Operator> '('<Measure_name>')'

<From_clause> ::= From <alias> in <XML_Element_Name>

(“.” <XML_Element_Name>)*(“,”<alias> in <XML_Element_Name>

(“.” <XML_Element_Name>)*)*

<By_clause> ::= By <alias>(',' <alias>)*

<AG_Fucnt> ::= COUNT | SUM | AVG | MAX | MIN

4.2 Opérateurs OLAP

Les auteurs de *(Ravat et al, 2006)* ont défini une série d'opérations algébriques, qui constitue un « noyau minimum fermé ». Dans cette série d'opérateurs, nous avons retenu les suivants :

- Opérateurs de forage : ***DrillDown***, ***RollUp***
- Opérateurs de rotation : ***DRotate***
- Opérateurs de structuration : ***Switch***, ***Nest***

Ces opérations ne sont pas explicitement définies dans notre langage, mais elles sont réalisables comme le montrent les exemples suivants.

Considérons la requête R1 définie à partir du StarCD de la figure 1. Cette requête calcule le nombre de mots clés pour chaque date de publication (paramètre PubliDate) :

```
R1:  Analyse count(k)
     From p in Paper,
          k in p.KeyWord,
          d in p.PubliDate
     By d
```

4.2.1 Opérateurs de forage

Les opérateurs de forage permettent d'effectuer des analyses en agrégeant ou en désagrégeant les données. Ils procèdent à un changement de niveau de granularité soit vers le haut (RollUp) soit vers le bas (DrillDown).

Exemple 2:

RollUp : Cet opérateur agrège les mesures en changeant le paramètre d'une dimension. La requête R2 compte ainsi le nombre d'auteurs référencés par mois :

```
R2:  Analyse count(k)
     From p in Paper,
          k in p.KeyWord,
          d in p.PubliDate,
          m in d.PubliMonth
     By m
```

DrillDown : Cet opérateur est l'inverse du précédent ; il augmente le niveau de détail des mesures. Ceci équivaut à réduire le niveau de profondeur du paramètre utilisé pour le calcul dans la clause From. La requête R3 compte à nouveau le nombre d'auteurs référencés par jour :

```
R3:  Analyse count(k)
     From p in Paper,
          k in p.KeyWord,
          d in p.PubliDate
     By d
```

Dans le cas des hiérarchies inversées, l'application de ces opérateurs est aussi inversée. La progression dans la hiérarchie depuis le paramètre de plus bas niveau correspond à l'opération DrillDown.

4.2.2 Opérateurs de rotation

L'opérateur de rotation ***DRotate*** permet de changer l'axe d'analyse (la dimension) utilisée pour le calcul de la requête. Ceci équivaut à changer le paramètre voulu dans la clause From de la requête.

Exemple 3:

La requête R4 calcule le nombre de mots clés par groupe d'auteurs (CoAuthors). Elle effectue une rotation entre les dimensions PubliDate et CoAuthors depuis la requête R3 :

```
R3:  Analyse count(k)
```

From p in Paper,

k in p.KeyWord,

co in p.CoAthors

By co

4.2.3 Opérateurs de structuration

Ces opérateurs permettent d'organiser le résultat d'une requête. Ainsi, l'opérateur Switch, qui effectue un classement, permute les valeurs d'un paramètre de dimension affiché ; ceci équivaut à changer l'ordre des variables dans la clause By. D'autre part, l'opérateur Nest, qui réalise une imbrication, permet d'imbriquer un paramètre dans un autre.

Exemple 4:

Considérons la requête R5 ci-dessous qui calcule le nombre d'articles par auteurs, par date de publication et par mots-clés (paramètres Author, PubliDate, et KeyWord) :

R5: Analyse count(p)

From p in Paper,

k in p.WordGroup.KeyWord,

a in p.CoAuthors.Author,

d in p.PubliDate

By a.FirstName, a.LastName, k, d

L'opérateur ***Switch*** permet de changer l'ordre d'affichage des variables a.FirstName et a.LastName, la clause By devient donc :

By **a.LastName**, **a.FirstName**, k, d

L'opérateur ***Nest*** permet d'imbriquer la variable d dans la variable k ; la clause By devient donc :

By a.LastName, a.FirstName, **k.d**

Ces classements et imbrications n'ont qu'un impact visuel ; les valeurs affichées restent inchangées pour le décideur, seule la structure de l'affichage change. La requête R5 devient au final :

R5: Analyse count(p)

From p in Paper,

k in p.WordGroup.KeyWord,

a in p.CoAuthors.Author,

d in p.PubliDate

By a.LastName, a.FirstName, k.d

Considérant que la structure d'un fichier XML peut se représenter sous la forme d'un graphe arborescent, le langage que nous présentons permet de naviguer dans cet arbre, et aide le décideur à situer la profondeur des éléments lors de la formulation des requêtes d'analyse. Ces requêtes une fois exprimées sont traduites de façon transparente en XQuery par le biais d'un traducteur. Elles sont ensuite appliquées à l'entrepôt.

5 Expérimentation

5.1 Processus de traduction

La figure 2 représente l'architecture globale du traducteur que nous avons développé. Celui-ci assure la traduction automatique des requêtes en XQuery et leur application sur l'entrepôt ; il rend ce processus entièrement transparent pour le décideur. Il prend en entrée la requête OLAP écrite par le décideur et s'aide du schéma multidimensionnel pour construire la requête XQuery équivalente suivant le processus suivant :

(*1*) Une fois la requête saisie par le décideur, le traducteur procède dans un premier temps à une analyse syntaxique en vérifiant que les mots clés saisis sont bien ceux attendus et sont placés au bon endroit dans la requête.

(*2*) La requête est ensuite transmise au module générateur de code qui procède d'abord à une analyse sémantique (vérification du typage et de la conformité à la grammaire BNF), avant de procéder à la traduction de la requête en s'aidant des informations issues du schéma de l'entrepôt.

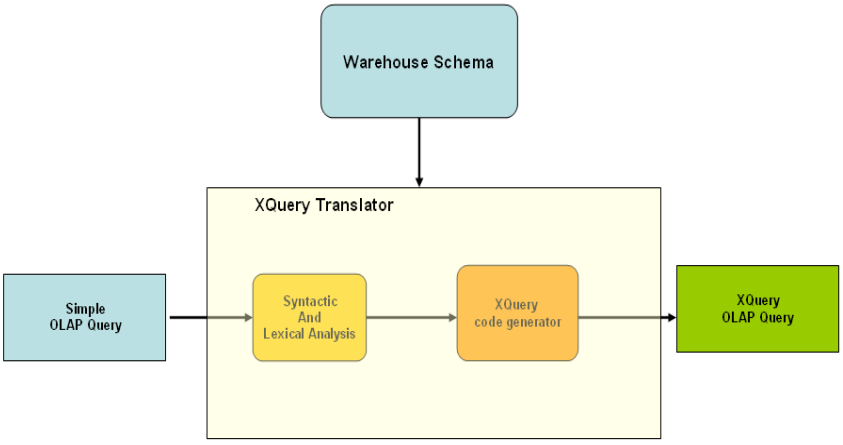

Figure 3. Architecture du traducteur

Nous avons défini une méthode de traduction qui peut se décrire informellement comme suit :

- ***Clause Analyse*** : Les mesures dans cette clause correspondent aux résultats à retourner. Elle seront déterminées dans le calcul de la clause ***let*** (qui permettra de déterminer les mesures à utiliser). Les mesures déterminées seront retrouvées dans la clause ***return*** du code XQuery accompagnées des fonctions d'agrégation associées. (Je ne comprends pas cette phrase).

- ***Clauses From*** : Pour chaque variable présente cette clause et aussi dans la clause **By**, il sera défini une clause ***for*** en XQuery, précisant elle aussi la profondeur des éléments ; le ***« distinct-values »*** en XQuery est utilisé si la variable n'a pas de fils dans l'arborescence.

  Chaque variable présente dans cette clause et aussi dans la clause Analyse, se retrouvera dans la clause ***return*** de la requête traduite.

- ***Clause By***: Les variables présentes dans cette clause ont plusieurs fonctions:

  - Elles permettent de préciser les dimensions (pour la définition des clauses ***for***)
  - Elles permettent de définir l'ordre d'affichage (clause ***order by***) si on travaille suivant plusieurs dimensions
  - Elles permettent d'appliquer une restriction sur les faits liés a la (aux) dimension(s) choisie(s) pour l'analyse (clause ***let***).

5.2 Comparaison de requêtes exprimées dans le nouveau langage et en XQuery

L'entrepôt matérialisé est alimenté à partir d'une collection d'articles scientifiques thématique, c'est à dire partageant la même structure (XSchéma). Il contient autant d'instances de fait que la source contient de documents. Les dimensions sont stockées dans des documents XML distincts, et sont liés au fait par des références comme on a pu s'en apercevoir dans la traduction de l'exemple 5. En effet nous avons séparé les instances de fait et les dimensions afin d'assurer l'unicité des valeurs des dimensions dans l'entrepôt, supprimant ainsi toute redondance.

L'entrepôt de documents est matérialisé dans la base de données eXist-DB qui est une base de données native XML, permettant une gestion aisée des collections XML par le biais du langage XQuery. Dans ce contexte, les requêtes exprimées par un décideur sont ensuite traduites en XQuery pour être appliquées à l'entrepôt, d'où l'importance du traducteur que nous avons développé. Nous montrons dans les exemples qui suivent quelques requêtes d'analyse ainsi que leurs traductions en XQuery réalisées par le traducteur :

Requête 1 : Calculer le nombre d'article publiés par auteur

Q1: Analyse count(p)

From p in Paper,
co in p.CoAuthors,
a in co.Author
By a.FirstName, a.LastName

Q1 traduite:

for $co **in** //DCoAuthors/CoAuthors,
$a **in** $co/Author
let $doc := //Paper[Dimensions/CoAuthors/ @ref=$co/@id]
return
<group>
< AuthorFirstName >{$a/FirstName}</AuthorFirstName>
< AuthorName >{$a/Name}</AuthorName>
<val>{count($doc)}</val>
</group>

Requête 2 : Calculer le nombre moyen de mot clés utilisé par chaque auteur

Q2: Analyse avg(count(k))

From p in Paper,
k in p.KeyWord,
co in p.CoAuthors,
a in co.Author
By a.FirstName, a.LastName

Q2 traduite:

for $co **in** //DCoAuthors/CoAuthors,
$a **in** $co/Author
let $doc := //Paper[Dimensions/CoAuthors/ @ref=$co/@id]/Measures/KeyWord
return
<group>
< AuthorFirstName >{$a/FirstName}</AuthorFirstName>

```
    < AuthorName>{$a/Name}</AuthorName>
    <val>{avg(count($doc))}</val>
</group>
```

Requête 3: Calculer le nombre d'article par mots clés et par années :

Q3: Analyse count(p)

```
    From p in Paper,
         w in p.WordGroup
         k in w.KeyWord
         m in p.PubliDate,
         y in m.PubliMonth.PubliYear
    By k, y
```

Q3 traduite:

```
for $w in //DWordGroup/WordGroup,
    $k in distinct-values($a/KeyWord),
    $m in //DPubliDate/PubliDate,
    $y in $c/PubliYear/@year

let $doc := //Paper[Dimensions/WordGroup/@ref = $a/@id
                    and Dimensions/PubliDate/@ref = $c/@id]
return
if (exists($doc))
then <group>
            <KeyWord>{$k}</KeyWord>
            <PubliYear>{data($d)}</PubliYear>
            <var>{count($doc)}</var>
    </group>
```

La condition if nous permet de filtrer sur les éléments $doc qui n'ont aucune valeur afin de ne pas surcharger l'affichage. Ces requêtes illustrent bien la simplicité d'expression d'une analyse OLAP dans notre langage par rapport à l'expression équivalente en XQuery. L'écart de forme de ces deux types de requêtes s'accroit avec la profondeur des éléments manipulés dans la structure hiérarchique des

documents. Par exemple, dans la requête 3, l'analyse se fait suivant deux dimensions. Dans la requête XQuery correspondante, on pourra constater la présence d'itérations marquées par la clause **for**. Cet aspect d'itération est simplifié dans le langage que nous proposons, grâce l'utilisation des variables définies dans la clause **From**. En effet celles-ci, par imbrication, permettent une navigabilité aussi aisée qu'en XQuery, tout en simplifiant l'expression de la requête. Le décideur bénéficie ainsi de toute la puissance de XQuery, en étant affranchi de sa complexité.

6 Conclusions et perspectives

Les systèmes OLAP actuels ne permettent pas le traitement et l'analyse des documents XML issus du Web. Or le format XML est devenu un standard pour le traitement et l'échange de données massives et hétérogènes.

Nous avons proposé dans cet article une approche de modélisation et d'analyse multidimensionnelle pour les entrepôts de documents XML natif. Nous avons défini un langage d'analyse OLAP pour les décideurs, qui s'applique sur un modèle multidimensionnel défini à l'aide du formalisme UML. Ce modèle permet de représenter la structure XML des documents à analyser à l'aide de faits, mesures et dimensions, facilitant l'expression des requêtes par les décideurs. Le langage permet aux décideurs de pouvoir exprimer des requêtes complexes à partir du schéma de l'entrepôt StarCD. Ces requêtes sont traduites en XQuery pour être appliquées sur un entrepôt de documents matérialisé. Nous avons à cet effet développé un module logiciel qui assure la traduction automatique des requêtes.

Nous développons actuellement un prototype basé sur l'approche définie dans cet article, et nous nous focalisons particulièrement sur les techniques d'intégration des données dans l'entrepôt à partir de la source des documents XML et du schéma (StarCD) de l'entrepôt. Nous cherchons également à optimiser le processus de traduction afin de gagner du temps sur le traitement des requêtes appliquées sur des masses de données et l'affinage des résultats. Il sera aussi question d'étudier notamment le temps de réponse lié à la traduction des requêtes. De même, l'adoption de ce langage passera probablement par le développement d'une interface homme-machine rendant plus ergonomiques, voire transparentes, les requêtes exprimées dans ce nouveau langage OLAP.

Références bibliographiques

Barry D. K., Cattell R.G.G. (1997). Titre du chapitre, *The Obkect Database Standard*, Morgan Kaufmann publisher, p. 83-115 (1997)

Beyer K., Chamberlin D., Colby L. S., Özcan F., Pirahesh H., Xu Y. (2005). Extending XQuery for Analytics, *Proceedings of the 2005 ACM SIGMOD international conference on Management of data*, p. 503-514

Golfarelli M., Rizzi S., Vrdoljak B (2001). Data warehouse design from XML sources, *Proceedings of the 4th ACM international workshop on Data warehousing and OLAP (DOLAP)*

Golfarelli M., Maio D., Rizzi S. (1998). The dimensional fact model: a conceptual model for data warehouses, *International Journal of Cooperative Information Systems 07*, 215-247.

Li Y. and An A. (2005). *Representing* UML Snowflake Diagram from Integrating XML Data Using XML Schema, *Proceedings of: 2005 International Workshop on Data Engineering Issues in E-Commerce (DEEC 2005)*, 9 April 2005, Tokyo, Japan

Nassis V., Rajagopalapillai R., Dillon T. S., Rahayu W. (2005). Conceptual and Systematic Design Approach for XML Document Warehouses, *Proceedings of the 2005 international conference on Computational Science and Its Applications*, p. 914-924.

Object Management Group (OMG): Unified Modeling Language (UML). *http://www.uml.org/*

Park B., Han H., and Song I. (2005). XML-OLAP: A Multidimensional Analysis Framework for XML Warehouses, *Data Warehousing and Knowledge Discovery*, Volume 3589, p. 32-42

Pérez J. M., Berlanga R., Aramburu M. J., and Pedersen T. B. (2008). Integrating Data Warehouses with Web Data: A Survey, Knowledge and Data Engineering, *IEEE Transactions on In Knowledge and Data Engineering*, IEEE Transactions on, Vol. 20, No. 7. (July 2008), p. 940-955

Pokorny J. (2001). Modelling Stars Using XML, *Proceedings of the 4th ACM international workshop on Data warehousing and OLAP (DOLAP)*

Rajesh Bordawekar and Christian A. Lang (2005). Analytical processing of XML documents: opportunities and challenges. SIGMOD Record, vol 34, pp. 27-32

Ravat F., Teste O., Tournier R., Zurfluh G. (2007). A Conceptual Model for Multidimensional Analysis of Documents, *Proceedings of the 26th international conference on Conceptual modelling*, p. 550-565

Ravat F., Teste O., Zurfluh G. (2006). Algèbre OLAP et langage graphique, *INFORSID*, p. 1039-1054 (2006)

Tseng, F. S. C., et A. Y. H. Chou (2006). The concept of document warehousing for multi-dimensional modelling of textual-based business intelligence. *Decision Support Systems (DSS)*, vol 42, Elsevier, pp. 727– 744.

Vrdoljak B., Banek M., and Rizzi S. (2003). Designing Web Warehouses from XML Schemas, *DaWaK 2003*, LNCS 2737, pp. 89-98

W3C Consortium (2013a). Extensible Mark-up Language (XML). *http://www.w3.org/XML/*

W3C-Consortium (2013b). Xml path language (Xpath) 3.0. http://www.w3.org/TR/xpath-30/

W3C-Consortium (2013c). Xml schema. http://www.w3.org/XML/Schema

W3C-Consortium (2013d). Xquery 3.0: An xml query language. http://www.w3.org/TR/xquery-30/