Knowledge Management by Querying Relational Views of XML Data: application to Microbiology

Damiano Migliori, Marie-Christine Rousset
LRI University of Paris Sud, France
{ migliori, mcr }@lri.fr

Ollivier Haemmerlé
UMR BIA INA P-G/INRA
Ollivier.Haemmerle@inapg.inra.fr

Abstract

In this paper we present a method for integrating and querying XML data in a relational setting. Though this method is generic, it has been motivated and validated by a knowledge management application on Microbiology: the e.dot project. The aim of the e.dot project was to enrich an existing relational database storing microbiological data dealing with food risk assessment with data resulting from a continuous Web technologic watch. The data coming from the Web are put in XML format and must be queried by the same relational query interface as the pre-existing relational database. The choice that we have made is to integrate new and possibly heterogeneous XML data by relational views over the schema of the existing database, called the Reference Schema. Those relational views are composed such that they form the so-called Global Relational Schema of the XML data. Microbiologist experts can then ask standard select-project-join queries, which combine the main query operators that are used in practice for extracting useful information from databases. We provide a query rewriting algorithm which decomposes a Global Query, which is a select-project-join query over the Global Relational Schema, into a set of local queries expressed in Xquery to be directly executable against the XML data. Compared to existing mediator approaches, the translation of join, select or project operations are pushed as much as possible into the XML queries, in order to optimize the execution of query plans in function of the available XML data.

Key Words: mapping techniques, mediator, mediated schema, rewriting queries using views, XML wrapping

1 Introduction

Since XML has become a standard for data exchange, more and more information in Knowledge Management applications will be described by XML documents combining text and data with a flexible structure described in DTDs or XML schemas. Though powerful specialized query languages exist for XML data (e.g. XQuery, which is recommended by W3C), they are difficult to use by end-users or practitioners, mainly because, in addition to being complex, they require the users to know exactly the structure (i.e. the DTD) of the documents they want to query. In an information integration setting [RR03, TS97, SYJ01], it is likely to have to deal with heterogeneous XML documents, i.e. documents corresponding to different DTDs, while being related to the same domain. Specialists of that domain want to ask their queries in terms of a single vocabulary, without having to express as many queries as DTDs. In the relational setting, select-project-join queries are widely used because they are easy to understand for users, while powerful enough to extract useful information from structured data. In particular, the microbiologists who use the e.dot relational database are familiar with such queries that they are used to pose through a user-friendly graphical interface called MIEL.

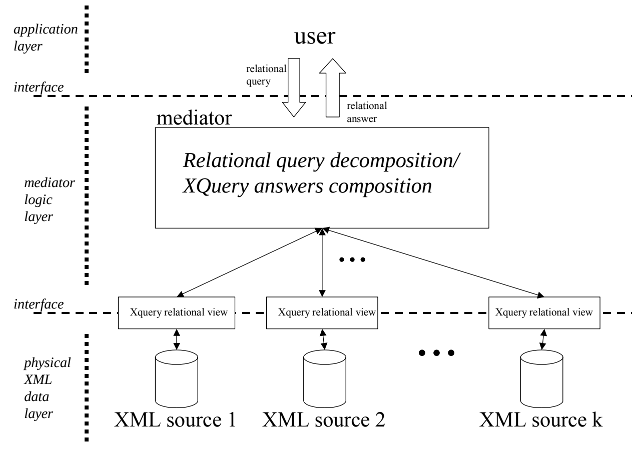


Figure 1 A top-view of the mediator

We have designed a generic information integration environment that permits to query in a relational way a collection of heterogeneous data coming from XML sources, giving the users the impression that they are interrogating a centralized relational system. In contrast with most existing works on transforming XML data into relational data (see [KKN03] for a survey), we do not materialize the XML data into relational views, but we define relational views that remain virtual: they are used to provide a relational schema to users who thus can query it by relational queries while the data remain stored in XML format possibly conform to heterogeneous DTDs. We define a relational view of an XML source corresponding to a given DTD by associating an XQuery query XV with a relation R(A1, …,An) of the relational Reference Schema. There can be several views XV1, …, XVk over the same given relation R(A1, …,An). A same XML document doc.xml (or DTD doc.dtd) can correspond to different relations R1,…., Rp: for each relation Ri, there exists an XV(Ri) query and all those XR(Ri) queries are executable against the same doc.xml (or doc.dtd). Our information integration system can be illustrated by Figure 1.

In Figure 2 we present two XML files conform to two different DTDs but related to the same domain (microbiology) and the relational view of the content of those documents in terms of the Relational Schema of reference of the e.dot application. It is important to point out that in our approach such views remain virtual.

document `lab-data2004.xml`

```
<LAB_DATA>
  <ANALYSIS_RECORD idNum="2003/01" >
    <CONDITIONS>
        <TEMPERATURE value="25 C" />
        <HUMIDITY value="80%" />
        <ATMOSPHERIC_PRESSURE value="1040" />
    </CONDITIONS>
    <TARGET>
      <RECEIPT idNum="01" name="Spaghetti Bolognese">
          <FOODCOMPONENT>
            <APPELLATION>Spaghetti</APPELLATION >
            <CATEGORY>Pasta</CATEGORY>
          </FOODCOMPONENT>
          <FOODCOMPONENT>
            <APPELLATION> Bolognese Meat </APPELLATION>
            <CATEGORY> Meat </CATEGORY>
          </FOODCOMPONENT>
          <FOODCOMPONENT>
            <APPELLATION> Tomato sauce </APPELLATION>
            <CATEGORY> Vegetables </CATEGORY>
          </FOODCOMPONENT>
          <MICROORGANISM_TRACES name="listeria monocytogenes"/>
          <MICROORGANISM_TRACES name="listeria innocua"/>
      </RECEIPT>
      ...
    </TARGET>
  </ANALYSIS_RECORD>

  <ANALYSIS_RECORD idNum="2004/08" >
    <CONDITIONS>
        <TEMPERATURE value="16 C" />
        ...
    </CONDITIONS>
    ...
  </ANALYSIS_RECORD>
  ...
</LAB_DATA>
```

document `catalog.xml`

```
<CATALOG>
  <YEAR>1999</YEAR>
  <FOODPRODUCT>
     <LABEL>Skimmed Milk</LABEL>
     <TEST idNum="01">
        <DURATION>Three days</DURATION>
        <TEMPERATURE>12 C</TEMPERATURE>
        <REPORT>
           <FACTOR>Z2</FACTOR>
           <BACTERIA>Listeria Seeligeri</BACTERIA>
           <COMMENT>It still tastes good</COMMENT>
        </REPORT>
     </TEST>
     <TEST idNum="02">
        <DURATION>One month</DURATION>
        <TEMPERATURE>33 C</TEMPERATURE>
        <REPORT>
           <FACTOR>YOK</FACTOR>
           <BACTERIA>Salmonella</BACTERIA>
           <COMMENT>It smells like my socks</COMMENT>
        </REPORT>
     </TEST>
     ...
  </FOODPRODUCT>
  <FOODPRODUCT>
     <LABEL>eggs</LABEL>
     <TEST idNum="01">
        <DURATION>Five days</DURATION>
        <TEMPERATURE>18 C</TEMPERATURE>
        <REPORT>
           <FACTOR>Y0Y</FACTOR>
           <BACTERIA>Cyclospora</BACTERIA>
           <COMMENT>Abnormal Colour</COMMENT>
        </REPORT>
     </TEST>
  </FOODPRODUCT>
...
</CATALOG>
```

FoodproductMicroorganismTemperature		
Tomato sauce	listeria monocytogenes	25 C
Bolognese Meat	listeria monocytogenes	25 C
Spaghetti	listeria monocytogenes	25 C
Tomato sauce	listeria innocua	25 C
Bolognese Meat	listeria innocua	25 C
Spaghetti	listeria innocua	25 C
…	…	…

MicroorganismAtmpressure	
listeria monocytogenes	1040
…	…

FoodproductMicroorganismTemperature		
skimmed milk	Salmonella	33 C
skimmed milk	Listeria Seeligeri	12 C
eggs	Cyclospora	18 C
…	…	…

Food-productFactor	
skimmed milk	Z2
eggs	Y0Y
…	…

e.dot Relational Schema of reference={ FoodproductPH , Food-productStructure , FoodproductMicroorganismTemperature, FoodproductFactor, MicroorganismAtmpressure }

(The real e.dot application Relational Scheme of reference is composed of 27 relation signatures)

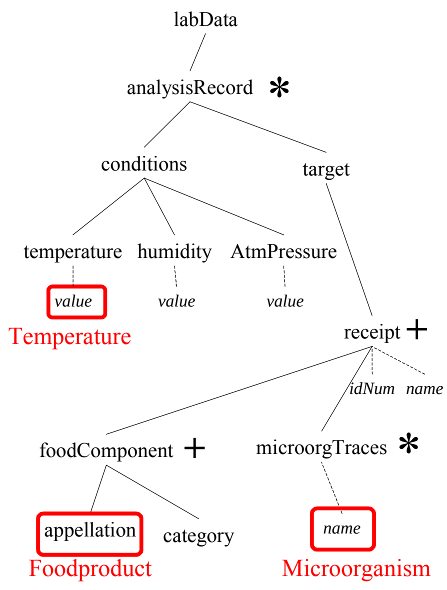
Figure 2

In Section 2, we describe the relational views of XML documents that we consider and we explain how XQuery queries are automatically generated from manual mappings provided by the administrator. In Section 3, we describe the relational queries that we consider. In Section 4, we describe the reformulation algorithm that we have implemented in order to transform a relational query into a union of XQuery queries directly executable against the available XML data. We conclude in section 5.

2 Mapping XML documents in relations of the Reference Schema

There are different classes of XML-to-relational mappings. *User-defined*: where the user specifies the mapping. *Generic*: fixed mappings, data and schema independent, like storing all the edges in a single table [FK99]. *Data-driven*: mappings inferred from data, mining the document looking for common regular patterns, building on such patterns. *Schema/DTD-driven*: using the DTD or the schema to decompose the document in tables [LWC01]. *Cost-based*: mapping inferred from schema, query workload and data [BFR02]. No solution is broadly better then the others: it depends strongly on the information system requirements. Our mapping is a trade-off between a *user-defined* and a *DTD-driven* mapping. We are given a set of XML documents (conform to some DTDs) and the relational Reference Schema RS. Mappings from a relation of RS to XML documents are expressed by queries defined in XQuery. The user defined aspect is to choose which DTD tree's nodes (N1,…, Nn) associate with a relation R(A1,…,An) in RS. We will not consider this aspect here, since it is out of the scope of this paper. Once associated the attributes A1,…,An to n nodes on the DTD tree (see figure 4 for an example), we automatically build the XQuery query referring to the DTD structure, ensuring the integrity of the representation of the hierarchical XML information in flat 1-to-1 tuples' relations.

XQuery is a standard XML query language [XQ05] elaborated by the W3C. The XQuery data model views an XML document as an ordered labelled tree. For navigating in a document, XQuery uses path expressions, whose syntax is borrowed from the abbreviated syntax of XPath [XP99]. The evaluation of a path expression on an XML document returns a list of information items, whose order is dictated by the order of the corresponding elements in the document. Typical query expressions of XQuery are FLWR expressions (for-let-where-return). Typically, a *let* binds a variable to a (path) expression, possibly nested *for* expressions make variables iterate over the result of (path) expressions, a *where* specifies restrictions on the variables, and the *return* constructs new XML elements as output of the query.



```
<!ELEMENT labData ( analysisRecord* )>

<!ELEMENT analysisRecord ( conditions, target )>
<!ELEMENT conditions( temperature, humidity,
          atmosphericPressure )>
<!ELEMENT temperature (#PCDATA)>
<!ELEMENT humidity (#PCDATA)>
<!ELEMENT atmosphericPressure (#PCDATA)>

<!ELEMENT target( receipt+ )>
<!ELEMENT receipt( foodComponent+, microOrganismTraces* )>

<!ELEMENT foodComponent( appellation, category )>
<!ELEMENT appellation (#PCDATA)>
<!ELEMENT category (#PCDATA)>

<!ELEMENT microOrganismTraces (#PCDATA)>
<!ATTLIST microOrganismTraces name CDATA #REQUIRED>

<!ATTLIST temperature value CDATA #REQUIRED>
<!ATTLIST humidity value CDATA #REQUIRED>
<!ATTLIST atmosphericPressure value CDATA #REQUIRED>
<!ATTLIST receipt
          idNum CDATA #REQUIRED
          name CDATA #REQUIRED>
```

labData.dtd

Mapping of the relation **FoodproductMicroorganismTemperature** on the DTD's tree, in *italic* the XML attribute leaves

Figure 3

Describing relational views using XQuery is an emerging approach that presents several positive features. We have chosen it as mapping language mainly for its declarative key aspect because separating the logic of the mapping from where and how it is processed makes the mediator flexible to the evolution of the information system. Furthermore the rich XQuery expressive power makes it possible to extend the general mapping lines defined in this paper to the special cases represented by an atypical use of XML syntax. The downside of using XQuery as mapping language is that at the execution time it could bring very poor results in terms of performance. In section 4 we describe in more details the criteria we adopt to improve performances.

A mapping from a relation R(A1,…,An) of RS to an XML document myDoc.xml conforming a DTD d is a query XV(R(A1,…, An),d) defined in XQuery as follows :

XV(R(A1,…, An) ,d):

```
1.  <TABLE>
2.      let $R := doc(myDoc.xml)/root
3.      for $L1 in $R/path1
4.        for $L2 in path2
5.          …
6.          for $Lk in pathk

7.          let $A1 in path'n
8.          …
9.          let $An in path'n

10.   return
11.     <TUPLE>
12.       <A1> $A1/text() </A1>
13.       <A2> $A2/text() </A2>
14.       ……
15.       <An> $An/text() </An>
16.     <TUPLE>
17. </TABLE>
```

An XQuery variable $Ai is associated with each attribute Ai from relation R(A1,…,An). The FOR clauses (lines 3-6) define how to navigate the document, the LET clauses (lines 7-9) bind the $Ai variables to the appropriate paths. The dependencies between the paths expressions `path1,...pathk, path'1,...,path'n` are defined in order to fix the context of the tuple we want to extract. When in a DTD Z there is a '*' or a '+' operator associated with a node Y, it means that a document valid on the DTD Z can present many nodes Y. Each subtree having Y as root node represents the formal context we want to preserve.

For example in the document `lab-data2004.xml` presented in Figure 2 (its DTD is in Figure 3), there are two `<ANALYSIS-RECORD>` nodes. The relation **FoodproductMicroorganismTemperature** is mapped on three nodes belonging to the subtree having `<ANALYSIS-RECORD>` as root (see Figure 3). We want all the triples of values in the tuples (Foodproduct, Microorganism, Temperature) belonging to the same sub-trees. Still referring to our example we consider as a result from a wrong mapping the tuple ("**Tomato sauce**", "**listeria monocytogenes**", "**16 C**") where the first two values come from the subtree `<ANALYSIS-RECORD idNum="`**`2003/01`**`">` and the third value, the temperature, from the subtree `<ANALYSIS-RECORD idNum="`**`2004/08`**`">`.

Such dependencies between paths are defined by visiting the DTD's tree. Once associated a node in the DTD's tree with each attribute in the relation R(A1,…,An), as in the example in figure 4, the mapping XV(R(A1,…,An),d) is generated automatically by applying the DTD tree recursive visit algorithm as follows:

DTD tree recursive visit algorithm:

```
BEGIN from the root node:
        VisitNode(root, root)

VisitNode(actual node , branching ancestor ):

        IF actual node is a '+' or a '*' node
                Add a for clause on path from
                branching ancestor to actual node;
                branching ancestor := actual node;

        IF actual node is associated with a relational
        attribute
                Add a let clause on path from branching
                ancestor to actual node;

        IF actual node is not a leaf
                FOR EACH child k of actual node
                        VisitNode(child k , branching
                        ancestor );
```

In Figure 4 there is an example of the result of the algorithm above used to produce the view XV(**FoodproductMicroorganismTemperature**, `labData.dtd`).

A view generated by the DTD tree recursive visit algorithm could present some redundancies due to the fact that the visit associates an XQuery FOR with each '+' or '*' node in the DTD tree. Considering the tree that represents the mapping visit strategy (see Figure 5), such redundancies can be eliminated respecting the rules described above. In Figure 5, the nodes "Branching-level-k" are related to the homonym XQuery variables in XV, and a continuous arch between two nodes B1 and B2 represents a FOR statement on variable V2 and a path depending on V1 from the view XV.

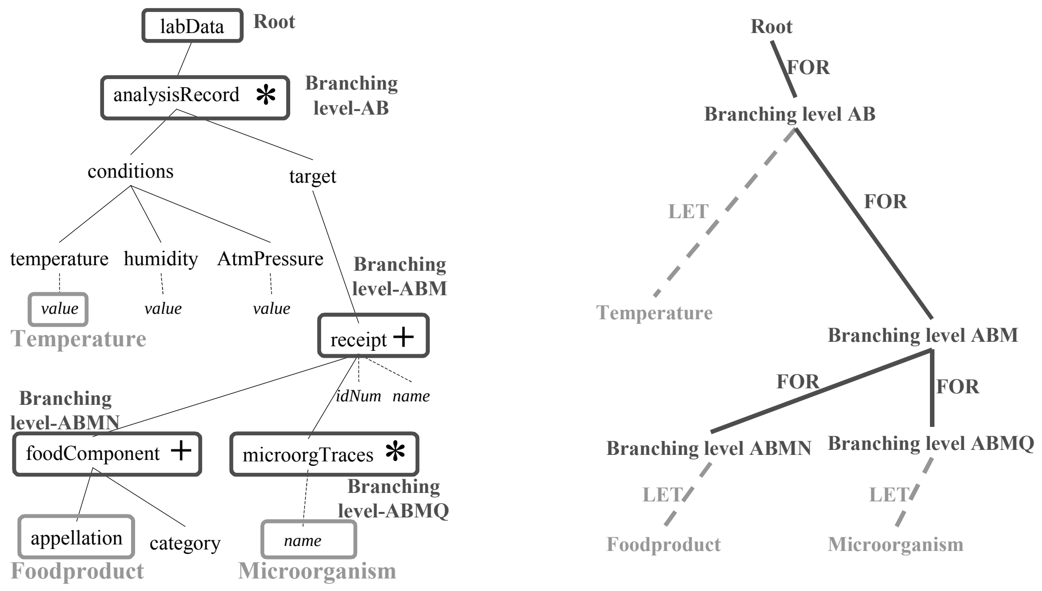
Rule 1
If between two nodes "Branching-level-i" and "Branching-level-j" there is a simple path composed by h nodes, all the FOR instruction composing that path can be replaced in XV by a single FOR instruction between "Branching-level-i" and "Branching-level-j".

Rule 2
If, given a node "Branching-level-i", in the subtree having "Branching-level-i" as root there is no LET arch, all the FOR instruction under "Branching-level-i" can be eliminated.

```
<TABLE>
{
let $Root := doc("archive2003.xml")/LAB_DATA

for $Branching-level-AB in $Root/ANALYSIS_RECORD
for $Branching-level-ABM in $Branching-level-AB/RECEIPT
for $Branching-level-ABMN in $Branching-level-ABM/FOODCOMPONENT
for $Branching-level-ABMQ in $Branching-level-ABM/MICROORGANISM_TRACES

  let $FoodProduct-COLUMN := $Branching-level-ABMN/APPELLATION/text()
  let $Microorganism-COLUMN := $Branching-level-ABMQ/@name/string()
  let $Temperature-COLUMN := $Branching-level-AB/CONDITIONS/TEMPERATURE/@value/string()

    return
    <TUPLE>
        <FOOD_PRODUCT>{ $FoodProduct-COLUMN }</FOOD_PRODUCT>
        <MICROORGANISM>{ $Microorganism-COLUMN }</MICROORGANISM>
        <TEMPERATURE>{ $Temperature-COLUMN }</TEMPERATURE>
    </TUPLE>
}
</TABLE>
```

An example of the view XV(**FoodproductMicroorganismTemperature,** `lab-data2004.dtd`). In bold the XQuery variables. Note that the mapping algorithm produces XQuery variables named like "**`$branching-level-ABMQ`**" where the string "**`ABMQ"`**" is easily interpretable as "there are three `FOR` in cascade from root node 'A' branching at nodes 'B', 'M' and 'Q' " (see Figure 5). This depends on the implementation of the algorithm. The letters A, B, etc… correspond to a numeration of the DTD's nodes in relation to the in-deep tree visit.

Figure 4



A representation of the mapping visit strategy on the DTD tree. On the left how the XQuery variables in the query XV are associated with the DTD tree nodes. On the right the dependences between paths in the FOR and LET statements in XV.

Figure 5

3 Queries over the Induced Relational Global Schema

In the previous section we have seen that given a set of XML documents, we can define a set V of relational views of some of those documents: a (possibly empty) set of relational views V(R)={XV(R(a1, …, an), f1), …, XV(R(a1, …, an), fm)} is associated with each relation R of the Reference Schema RS. The Global Schema induced by those views is a subset of the Reference Schema RS defined as follows.

Definition (Induced Global Schema)

Let RS be a relational Reference Schema and V be a set of relational views over RS of a set of documents. The Global Schema induced by V, denoted R(V,RS), is the set of relations from RS having a non-empty associated set of relational views.

In other words, R(V,RS) is composed of all the relations of RS which have been mapped in at least one XML document. For example, the induced global schema corresponding to the views on XML documents presented in Figure 2 is:

R(V,RS) = { FoodproductMicroorganismTemperature, FoodproductFactor, MicroorganismAtmpressure }

The induced global schema that is composed of the relations of interest for the user of the XML base is the "relational point of view" that the user has on the XML base. It is presented to the user by means of a graphical user interface (see Figure 6).

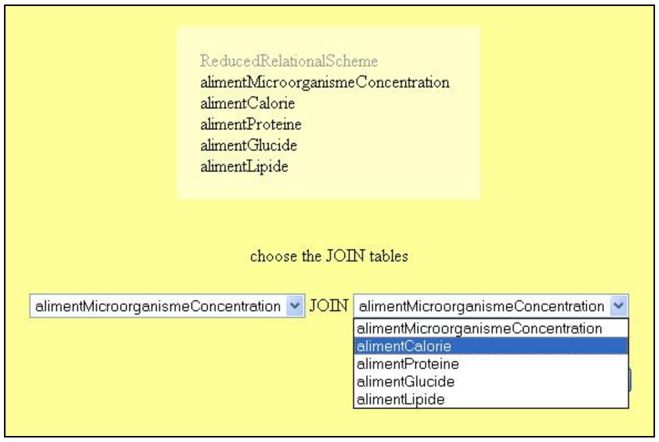


Figure 6

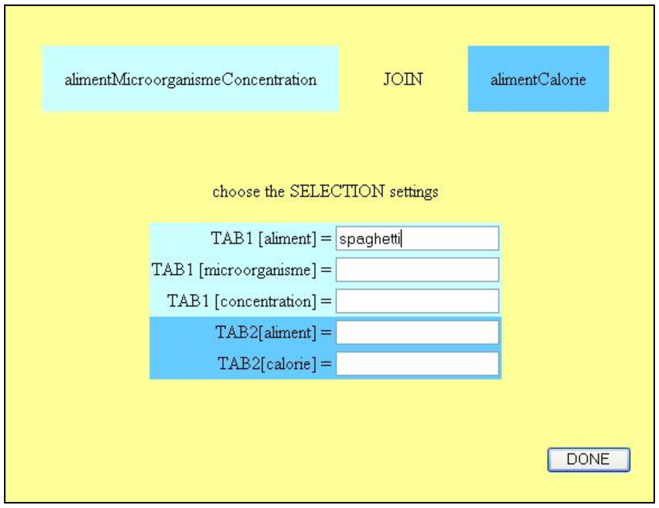
We propose to query the induced Global Schema by means of a standard join-selection-projection query language. To be more precise, we propose to use the following operations of the relational algebra:

given RA(A1, …, An) ∈ R, RB(B1, …, Bm) ∈ R(V,RS)

- RA **Join** RB on condition (Ai = Bj) for a given couple i,j
- **Projection** RA on {A'} subset of { A1, …, An}
- **Selection** RA on (Boolean condition on Ak)

In the current version of the application, a Join is possible between two tables and the user can set up multiple selection clauses on equi-conditions. The queries are expressed by the user through a graphical user interface (see Figure 7) that passes to the underlying mediator the following elements:

- The two join operand relations RA(A1, …, An) and RB(B1, …, Bm) chosen among those in the Induced Global Schema
- The list of attributes to project among {A1, …, An,B1, …, Bm}
- The (eventually empty) list of selection conditions.


alimentMicroorganismeConcentration
JOIN
alimentCalorie
choose the SELECTION settings
DONE

TAB1 [aliment] =	spaghetti
TAB1 [microorganisme] =	
TAB1 [concentration] =	
TAB2[aliment] =	
TAB2[calorie] =	

Figure 7

The next section presents the way a relational query over the induced Global Schema is reformulated into a union of XQuery queries, whose execution using an XQuery engine provides the complete set of answers of the initial relational query.

4 Reformulation and evaluation of a relational query in XQuery

Using XQuery to map the XML documents to the relational views is very useful in terms of logical independence of the mediator from the XML sources, but it could easily bring to nested queries that tend to alter the system's efficiency. In addition it is important to pay attention in executing navigational queries over very large amount of data because it can be critical even in native XML repository that adopt sophisticated indexing techniques. Our query-reformulation algorithm tries to respond to these performance requirements in two ways. We decompose the Global query in the union of several local queries that can be executed in parallel. Each local query involves locally no more than two documents with a significant optimisation in the context of a native XML repository. Moreover every local query does not present nested queries.

While the user can access only the Induced Global Schema, the mediator keeps an internal representation on how the Induced Global Schema is built on a composition of Local Views, as in Figure 8. In addition it keeps information on how each relation R from the Induced

Global Schema is mapped in views XV(R,s)[1] on each source s where R has been mapped. Referring to figure 6.A it means that, for example, the mediator knows how to map the relation RelA on the XML sources 1,2 and 4 in XQuery.

Induced Global Schema	**RelA**	**RelB**	**RelC**	…	**Relβ**
XML source s1	XV(RelA,s1)		XV(RelC,s1)		
XML source s1	XV(RelA,s1)		XV(RelC,s1)		
XML source s1		XV(RelB,s3)	XV(RelC,s1)		
XML source s1	XV(RelA,s1)		XV(RelC,s1)		XV(Relβ,s4)
…					
XML source sn		XV(RelB,s3)			XV(Relβ,s4)

A representation the way the Induced Global Schema is built on a composition of local views.

Figure 8

The Global extension of a relation from the Induced Global Schema corresponding to the set of tuples that can be extracted from the available XML data is defined as follows:

Definition (Global Extension).
Let

R(A1, …, An) ∈ R(V,RS) be a relation of the Global Schema induced by a set V(R(A1, …, An)) of relational views over the reference schema RS, where
V(R(A1, …, An)) = {XV1(R(a1, …, an),d1), …, XVm(R(a1, …, an),dm)}

The Global Extension GE(R) of R is the set of tuples :

$$GE(R)=\bigcup_{(1 \le i \le m)} exec(XV(R(a1, \dots, an), di),$$

where exec(XV(R(a1, …, an)), di) is the set of tuples resulting from the execution of the queries in XQuery defining the mapping XV on the document di.

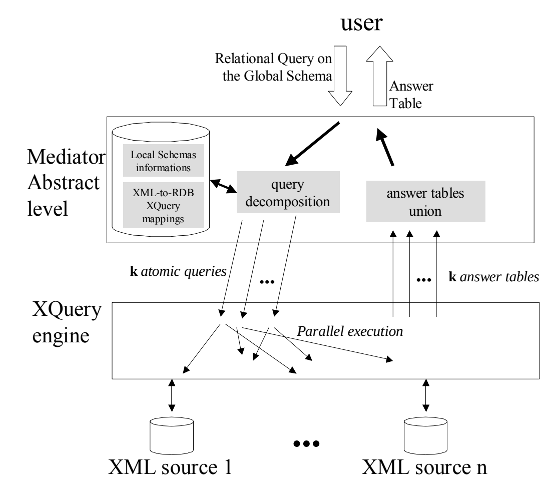
The Query Decomposition Algorithm is the core of the information integration system and the one we propose here makes it possible to produce joins between data coming from different XML sources. Considering what we already said we can decompose a join operation (**RelA** *Join* **RelB**) defined by the user by a relational query on the Induced Global Schema into the operation on the *global extensions* (GE(RelA) *Join* GE(RelB)), which are sets of tuples, as follows:

$$(GE(RelA)\ \textit{Join}\ GE(RelB)) =$$
$$= (\bigcup_{(1 \le i \le m)} exec(XV(RelA, di))$$
$$\textit{Join}\ (\bigcup_{(1 \le j \le m)} exec(XV(RelB, dj))$$

in view of the fact that a join operation is a Cartesian product on two collections of tuples plus a selection we can produce the subsequent equivalent decomposition:

$$(GE(RelA)\ \textit{Join}\ GE(RelB))=$$
$$=\bigcup_{\mathbf{i},\mathbf{j}} (exec(XV(RelA,s\mathbf{i})\ \textit{Join}\ exec(XV(RelB,s\mathbf{j}))$$

The initial relational query (**RelA** *Join* **RelB**) defined by the user on the Induced Global Schema, (and consequently to be executed on the whole set of XML sources) is now reformulated in the union of several XQuery queries which accomplish the (exec(XV(RelA,si) *Join* exec(XV(RelB,sj)) basic operation over no more than two sources; we call such queries *atomic join queries*. (See Query 3 for an example)



The mediator internal functioning. The user's Relational Query over the Global Schema produces **k** XQuery atomic-queries like the one in Query 4. A parallel execution of the **k** atomic-queries returns **k** sets of tuples (called answer tables). The **k** sets of tuples are put together forming the answer table returned to the user.

Figure 9

The execution of the **k** atomic queries generated from the initial relational query over the Global Schema generate **k** sets of matching tuples; these **k** sets of tuples are put together by the mediator and returned to the user as an answer table. The mediator internal functioning is summed up in Figure 9.

[1] A view of the Relation R on a XML source s XV(R,s) is analogue to a view of the Relation R on a document f XV(R,f) considered in section 2, considering the XML source equivalent to an XML document.

The execution of an atomic join query involves two nested queries: exec(XV(RelA,si)) and exec(XV(RelB,sj)) with an undesired temporary materialization of the Relational Views XV(RelA,si) and XV(RelB,sj) (example in Query 2 and in Query 3). Using the XQuery equivalence rules, such a query can be easily rearranged in an equivalent query where there is no temporary materialization of the Relational Views XV. We use the same navigational statements of the relational views XV(RelA,si) and XV(RelA,sj) (an example in Query 2 lines 3-10, Query 3 lines 3-7) directly in the navigational context of a single *optimized* atomic-join-query (see Query 4). Our mediator implementation starting from a relational query on the Global Schema produces directly a set of k *optimized* atomic queries with no nested queries, that means materializing only the tuples returned to the user in the answer table.

An atomic join query can also take care of the projection and selection specifications of the relational query set up by the user by: 1) adding the selection Boolean condition in and-cascade to the join condition into the WHERE clause (line 16 in the Query 4). 2) Choosing which attributes return in the RETURN clause (lines 19-22 in the Query 4).

```
<TABLE>
{
let $TAB1 := doc(XV(FoodproductMicroorganismTemperature, lab-data2004.xml))/TABLE
for $TAB1-tuple in $TAB1/TUPLE
 let $TAB1-FoodProduct-COLUMN := $TAB1-tuple/FOODPRODUCT
 let $TAB1-Microorganism-COLUMN := $TAB1-tuple/MICROORGANISM
 let $TAB1-Temperature-COLUMN := $TAB1-tuple/TEMPERATURE

let $TAB2 := doc(XV(FoodproductFactor, catalog.xml))/TABLE
for $TAB2-tuple in $TAB2/TUPLE
 let $TAB2-FoodProduct-COLUMN := $TAB2-tuple/FOODPRODUCT
 let $TAB2-Factor-COLUMN := $TAB2-tuple/FACTOR
the join condition:
where ($TAB1-FoodProduct-COLUMN = $TAB2-FoodProduct-COLUMN )

 return
<TUPLE>
  <FOODPRODUCT>{ $TAB1-FoodProduct-COLUMN }</FOOD_PRODUCT>
  <MICROORGANISM>{ $TAB1-Microorganism-COLUMN }</MICROORGANISM>
  <TEMPERATURE>{ $TAB1-Temperature-COLUMN }</TEMPERATURE>
  <FACTOR>{ $TAB1-Factor-COLUMN /string()}</FACTOR>
    </TUPLE>
}
</TABLE>
```

Query 1 - atomic join query (exec(XV(RelA,si) *Join* exec(XV(RelA,sj))

```
1.      <TABLE>
2.      {
3.      let $TAB1-Root := doc("lab-data2004.xml")/LAB_DATA
4.      for $TAB1-lev-AB in TAB1-$Root/ANALYSIS_RECORD
5.      for $TAB1-lev-ABM in $TAB1-level-AB/RECEIPT
6.      for $TAB1-lev-ABMN in $TAB1-level-ABM/FOODCOMPONENT
7.      for $TAB1-lev-ABMQ in $TAB1-level-ABM/MICROORGANISM_TRACES
8.       let $TAB1-FoodProduct-COLUMN := $TAB1-lev-ABMN/APPELLATION/text()
9.       let $TAB1-Microorganism-COLUMN := $TAB1-lev-ABMQ/@name/string()
10.      let $TAB1-Temperature-COLUMN := $TAB1-lev-AB/CONDITIONS/TEMPERATURE/@value/string()

11.     return
12.     <TUPLE>
13.     <FOODPRODUCT>{ $TAB1-FoodProduct-COLUMN }</FOOD_PRODUCT>
14.     <MICROORGANISM>{ $TAB1-Microorganism-COLUMN }</MICROORGANISM>
15.     <TEMPERATURE>{ $TAB1-Temperature-COLUMN }</TEMPERATURE>
16.     </TUPLE>
17.     }
18.     </TABLE>
```

Query 2 - view XV(FoodproductMicroorganismTemperature, lab-data2004.xml)

```
<TABLE>
{
let $TAB2-Root := doc("catalog.xml")/CATALOG
for $TAB2-lev-AC in $TAB1/FOODPRODUCT
for $TAB2-lev-ACE in $TAB1-lev-AC/TEST
  let $TAB2-FoodProduct-COLUMN := $TAB2-lev-AC/LABEL
  let $TAB2-Factor-COLUMN := $TAB2-lev-ACE/REPORT/FACTOR

return
  <TUPLE>
  <FOODPRODUCT>{ $TAB1-FoodProduct-COLUMN /text()}</FOODPRODUCT>
  <FACTOR>{ $TAB1-Factor-COLUMN /string()}</FACTOR>
  </TUPLE>
}
</TABLE>
```

Query 3 - view XV(`FoodproductFactor`, `catalog.xml`)

```
<TABLE>
{
mapping  for relation FoodproductMicroorganismTemperature on source lab-data2004.xml:
let $TAB1-Root := doc("lab-data2004.xml")/LAB_DATA
for $TAB1-lev-AB in TAB1-$Root/ANALYSIS_RECORD
for $TAB1-lev-ABM in $TAB1-level-AB/RECEIPT
for $TAB1-lev-ABMN in $TAB1-level-ABM/FOODCOMPONENT
for $TAB1-lev-ABMQ in $TAB1-level-ABM/MICROORGANISM_TRACES
  let $TAB1-FoodProduct-COLUMN := $TAB1-lev-ABMN/APPELLATION/text()
  let $TAB1-Microorganism-COLUMN := $TAB1-lev-ABMQ/@name/string()
  let $TAB1-Temperature-COLUMN := $TAB1-lev-AB/CONDITIONS/TEMPERATURE/@value/string()
mapping  for relation FoodproductFactor on source file catalog.xml
let $TAB2-Root := doc("catalog.xml")/CATALOG
for $TAB2-lev-AC in $TAB1/FOODPRODUCT
for $TAB2-lev-ACE in $TAB1-lev-AC/TEST
  let $TAB2-FoodProduct-COLUMN := $TAB2-lev-AC/LABEL
  let $TAB2-Factor-COLUMN := $TAB2-lev-ACE/REPORT/FACTOR

the join condition:
where ($TAB1-FoodProduct-COLUMN = $TAB2-FoodProduct-COLUMN )

return
<TUPLE>
<FOODPRODUCT>{ $TAB1-FoodProduct-COLUMN }</FOOD_PRODUCT>
<MICROORGANISM>{ $TAB1-Microorganism-COLUMN }</MICROORGANISM>
<TEMPERATURE>{ $TAB1-Temperature-COLUMN }</TEMPERATURE>
<FACTOR>{ $TAB1-Factor-COLUMN /string()}</FACTOR>
</TUPLE>
}
</TABLE>
```

Query 4 - optimized atomic join query (XV(RelA,si) *Join* XV(RelA,sj))

5 Conclusion

The method described in this paper for integrating and querying XML data through relational views has been implemented in the setting of the e.dot project. The e.dot project aimed at enriching an existing relational database (called Sym'Previus) dealing with predictive microbiology with XML data extracted from the Web.

The Sym'Previus database [BHT03] is being developed since 1999, in order to gather data concerning the microbiological risk in food products. Such a database is of large interest for the governmental institutions as well as the food industry, since it can help them to understand the previous safety problems and to prevent new crisis.

The Sym'Previus database contains about 10.000 pieces of information extracted manually from the scientific bibliography in microbiology, but also given by the industrial partners of the project. That base is accessible through a Web interface to the Sym'Previus partners and subscribers who query it by means of a relational-like language.

One of the specificities of the Sym'Previus database is its incompleteness, since the number of experiments involving each bacterium with each food product in every experimental condition is potentially infinite. So a way of complementing the database with data automatically found on the Web as it is proposed in the e.dot project is a real asset for such a database. The integration of the data coming from the relational database and the XML documents coming from the Web by means of a relational-like query language is a point of interest of our approach since the microbiologists already know that interface.

Many researchers have studied the problem of storing XML documents into relational tables [BFR02, ADF04, BFR02, FK99], and also the converse problem of exporting relational data into XML [HJLPM04, FFHS02]. The practical motivation of the former problem is that native XML storage and querying technologies are still too young to offer performances and robustness comparable to the mature DBMS systems. The practical motivation of the latter problem is that XML is becoming the standard format for exchanging data. Our work is at the confluence of those two lines of work. It makes cohabit nicely the two data models by combining their respective advantages: the relational data model is exploited for its logical simplicity thus providing a simple and synthetic query interface for end-users while the XML format is exploited for extracting and integrating possibly heterogeneous data coming from the Web.

In our current work, the instances of the relational views of XML documents are atomic textual data (strings at the leaves of the trees representing the queried XML documents). We plan to extend our work to allow that relational queries over XML documents possibly deal with tree-structured fragments of XML documents.

References

[ADF04] Sihem Amer-Yahia, Fang Du, Juliana Freire. A comprensive solution to the XML-to-Relational Mapping Problem. In *Proceedings of WIDM*, 2004

[BFR02] P. Bohannon, J. Freire, P. Roy, J. Simeon. From XML Schema to Relations: A Cost-Based Approach to XML Storage, In *Proc. of Intl. Conf. on Data Engineering (ICDE)*, 2002.

[BHT03] Patrice Buche, Ollivier Haemmerlé, Rallou Thomopoulos. Integration of heterogeneous, imprecise and incomplete data: an application to the microbiological risk assessment. . *In Proceedings of the 14th International Symposium on Methodologies for Intelligent Systems, ISMIS'2003, Maebashi, Japan, October 2003, Lecture Notes in* AI #2871, Springer, pp. 98-107.

[FFHS02] Catalina Fan, Jhon Funderburk, Hou-in Lam, Jayvel Shanmugasundaram. XTABLES: Bridging Relational Tecnology and XML. *IBM Systems Journal*, 2002

[FK99] D. Florescu, D. Kossman. Storing and Querying XML data using an RDMBS, *IEEE Data Engineering Bulletin*, 22(3):27–34, 1999.

[HJLPM04] Alan Halverson, Vanja Josifovski, Guy M. Lohman, Hamid Pirahesh, Mathias Mörschel: ROX: Relational Over XML. *VLDB 2004*: 264-275

[KKN03] R. Krishnamurthy, R. Kaushik, J. F. Naughton. XML-to.SQL Query Translation Literature: The state of the Art and Open Problems. *In XML Database Symposium*, 2003.

[LWC01] D. Lee, Wesley W. Chu. Constraints-Preserving Inlining Algorithm for Mapping XML DTD to Relational Schema, J. Data & Knowledge Engineering (DKE), 39(1):3{25, Oct. 2001.

[RR03] Marie-Christine Rousset, Chantal Reynaud. Knowledge Representation for Information Integration. *Information Systems International Journal, Special issue: Web-Universal Integration,* Volume 29, Number 1, p. 3-22.

[SYJ01] H eekyoung Seo, Jaeyoung Yang, Joongmin Choi. Knowledge-based Wrapper Generation by Using XML. *IJCAI-2001 Workshop on Adaptive Text Extraction and Mining (ATEM 2001),* pp. 1-8, Seattle, USA, 2001.

[TS97] Mary Tork Roth, Peter Schwarz. Don't Scrap it, Wrap it! A Wrapper Architecture for Legacy Data Sources. In *Proceeding of the 23th VLDB Conference, Athens*, pp. 266-275, Greece, 1997.

[XP99] J. Clark and DeRose, XML Path Language (XPath), version 1.0, W3C Recommendation, http://www.w3.org/TR/xpath, November 1999

[XQ05] D. Chamberlin, D. Florescu, J. Robie, J. Simeon and M. Stefanescu, XQuery: A Query Language for XML, W3C Working Draft, http://www.w3.org/TR/xquery February 2005.