Enriching a Relational Data Warehouse by Integrating XML Data: Report on the e.dot Project Applied to Microbiology

Helene Gagliardi[2], Ollivier Haemmerlé[1], Damiano Migliori[2], Nathalie Pernelle[2], Marie-Christine Rousset[2] and Fatiha Saïs[2]

[1] UMR MIA INA P-G/INRA 16 rue Claude Bernard,F-75231 Paris Cedex 05, France
[2] Université de Paris-Sud, LRI, Bâtiment 490,
F-91405 Orsay Cedex, France

Abstract. In this paper we present two methods for integrating (and querying) data in a relational setting. These methods have been motivated and validated by a knowledge management application on Microbiology, the edot project. The aim of e.dot project is 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 interface as the pre-existing relational database.
The first method allows to integrate possibly heterogenous XML data by relational views over the schema of the existing database. These relational views are not materialized but we provide a query rewriting algorithm which decompose a select-project-join query into a set of local queries expressed in Xquery.
The second method focus on the data tables contained in the documents found on the Web. The method takes advantage of the presence of relational structures to automatically transform these data in order to make them compatible with the schema of the existing database . Those tables are semantically enriched by means of tags and values coming from the ontology of the application.

1 Introduction

Our work deals with the automatic construction of domain specific data warehouses. More precisely, our goal is to integrate automatically information found on the Web with existing information stored in different databases.

Our application domain concerns the microbiological risk in food products. In order to understand and to prevent such risks, the Sym'Previus project has been launched by French governmental institutions. During the Sym'Previus project, the MIEL++ system has been built [Buche *et al.*2004]. MIEL++ is a tool based on a database, containing experimental results and industrial results about the behaviour of pathogenic germs in food products depending on several parameters, such as the temperature, the pH, etc. The Sym'Previus database is incomplete by nature since the number of possible experiments is potentially infinite.

The work presented in this article takes place within the e.dot project, which is a cooperation between the INA P-G/INRA MIA group, the Xyleme start-up, the IASI-Gemo team (LRI) and the Verso-Gemo team (INRIA-Futurs). The goal of the e.dot project is to palliate the incompleteness of the database by complementing it with data automatically extracted from the Web. The drawback of such a technique is that the way the data are expressed on the Web is very heterogeneous. For example, the terms used in the scientific articles in microbiology can be different from an article to another. A way of solving that heterogeneity issue can be to query the existing database and the Web documents through a mediated architecture based on a domain ontology.

Two methods have been adjusted in order to make possible the query processing on the data extracted from the Web.

The first one proposes 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). It is important to point out that in this method such views remain virtual.

The second method propose to translate the data in order to make them compatible with the Sym'Previus ontology used in the mediated schema. This method exclusively focus on documents in Html or Pdf format which contain data tables; actually data tables are very common presentation scheme for authors in order to describe experimental results, statistical or other synthetic data in scientific articles. In our system, these tables are extracted and transformed in a generic XML representation called XTab. These documents are then semantically enriched and stored in the data warehouse. In our approach, we want this transformation to be as automatic and flexible as possible, only driven by the ontology and the way the data have been structured in the original table. Thus, we have defined a Document Type Definition named SML (Semantic Markup Language) which can automatically be generated using the relational Reference Schema of the ontology and which can deal with additional or incomplete information in a semantic relation, ambiguities or possible interpretation errors. This approach has been implemented and tested on real data from the e.dot project.

The paper is structured as follows. In Section 2, we present the first method. We describe first 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. Then, we describe the relational queries that we consider. Finally, 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.

In Section 3, we present the second method. We first introduce the XTab format, the Sym'Previus ontology, and a simple example in order to explain the aims of the semantic enrichment task. Then we introduce the way we identify the ontology terms represented by the columns of a table. We also present the identification of semantic relations in data table and explain the instantiation of such semantic relation. We then give an idea of the possible use of the semantic enrichment during the query processing. Finally, some experimental results are shown.

In the conclusion, we present related works and we give future directions for both methods.

2 Information integration by querying relational views of XML data

This first method allows to integrate heterogeneous XML data by relational views over the schema of the existing database. This 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.

2.1 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 [Florescu & Kossman1999]. *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 [Lee & Chu2001]. *Cost-based:* mapping inferred from schema, query workload and data [Bohannon *et al.*2002]. 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

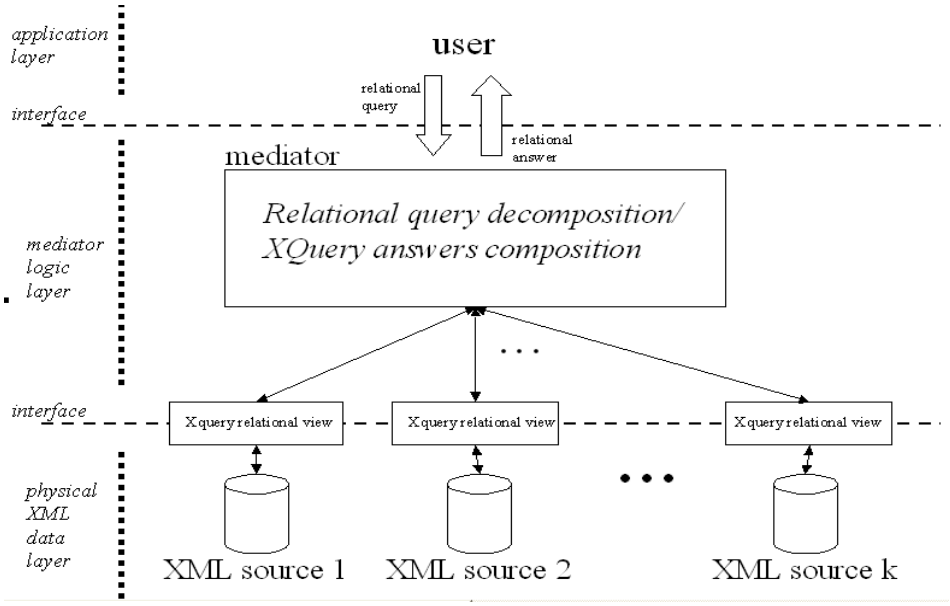


Fig. 1. A top-view of the mediator

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 [Chamberlin *et al.*2005] 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 [Clark & DeRose1999]. 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.

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

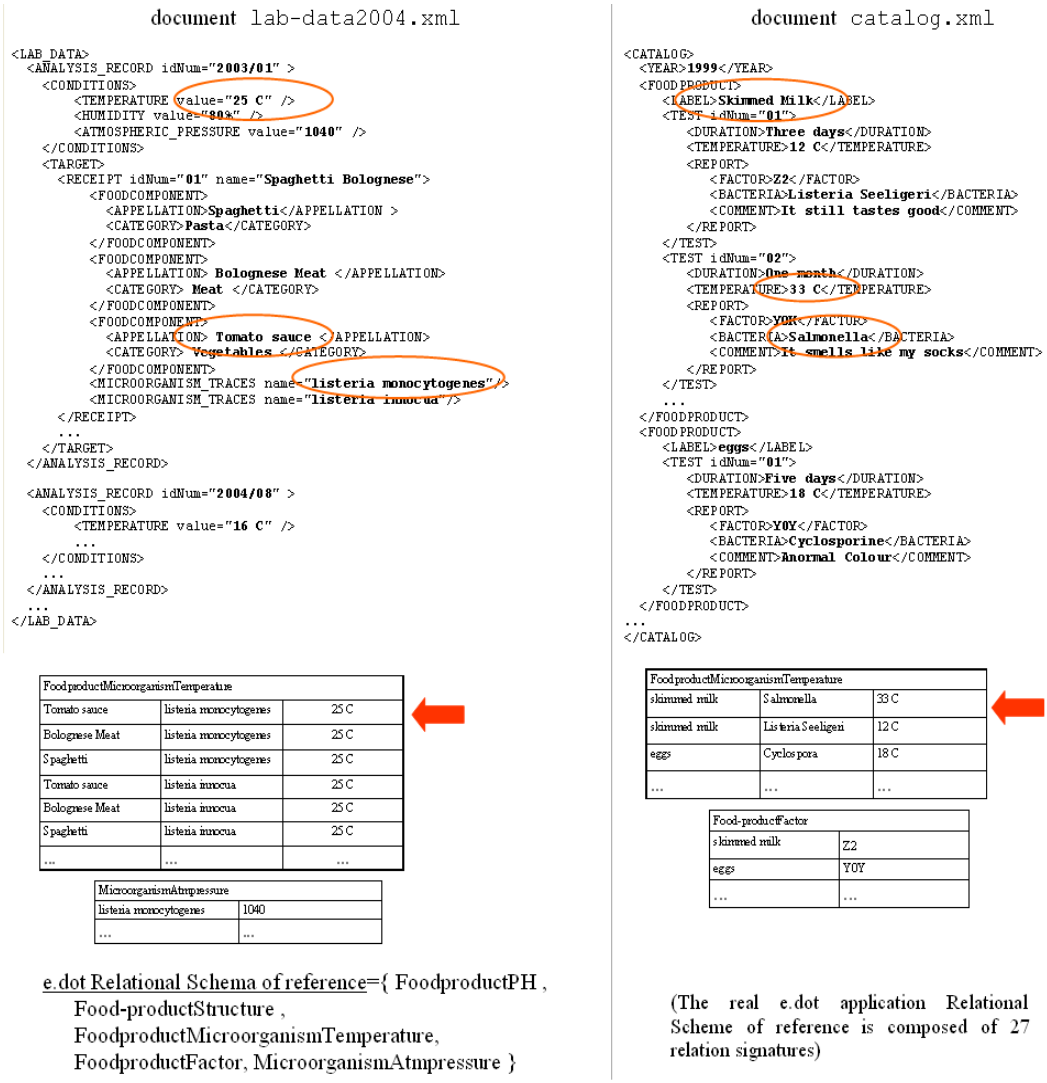


Fig. 2.

mapping language is that at the execution time it could bring very poor results

in terms of performance. In section 2.3 we describe in more details the criteria we adopt to improve performances.

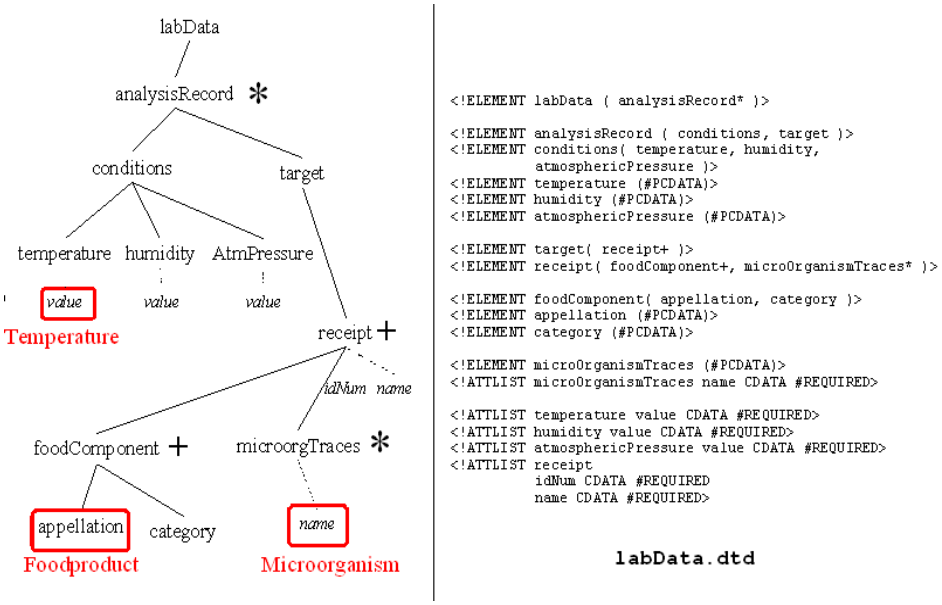


Fig. 3.

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 :

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)*

```
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>
```

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 9, 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;

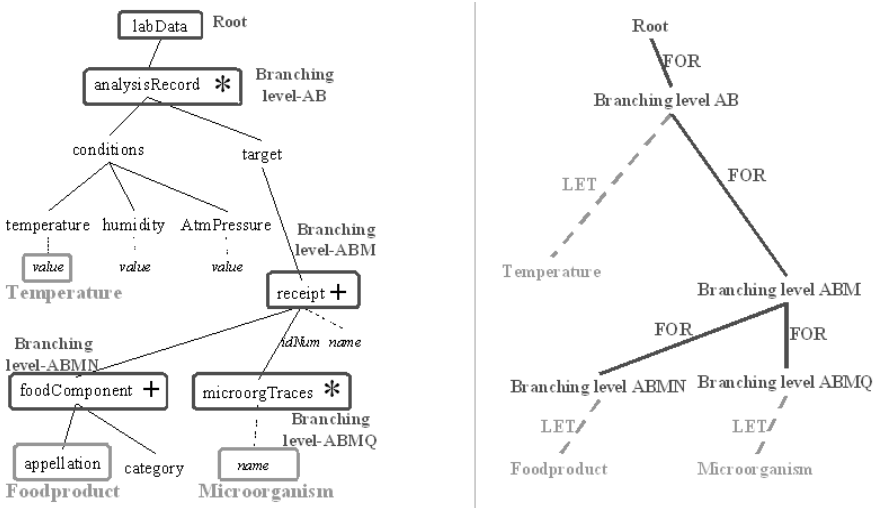
*IF **actual node** is associated with a relational attribute*
Add a let clause on path from branching ancestor to actual node;

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

In Figure 9 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 4), such redundancies can be eliminated respecting the rules described above. In Figure 4, 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.



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.

Fig. 4.

2.2 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 1. *(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 5-interface A).

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 5-interface B) 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.

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.

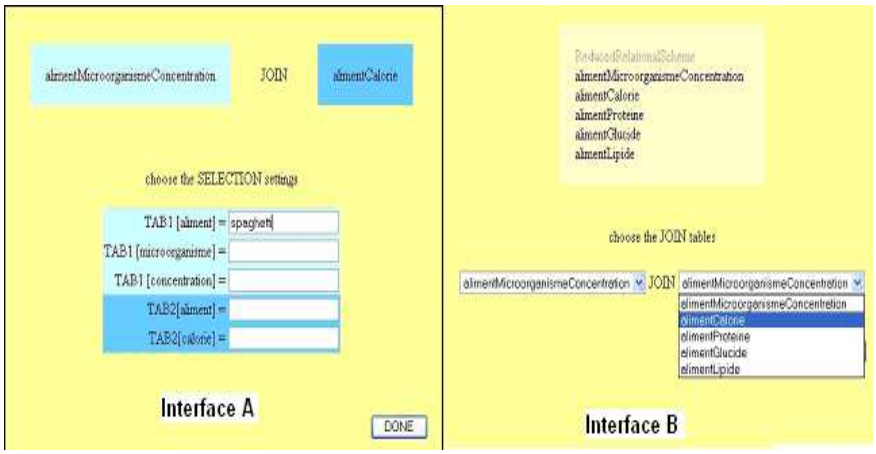


Fig. 5. Two GUI screenshots

2.3 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 6. In addition it keeps information on how each relation R from the Induced Global Schema is mapped in views XV(R,s)[3] 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.

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:

[3] 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.

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(Rel,s4)

Fig. 6. *A representation the way the Induced Global Schema is built on a composition of local views.*

Definition 2. *(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 :

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

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:

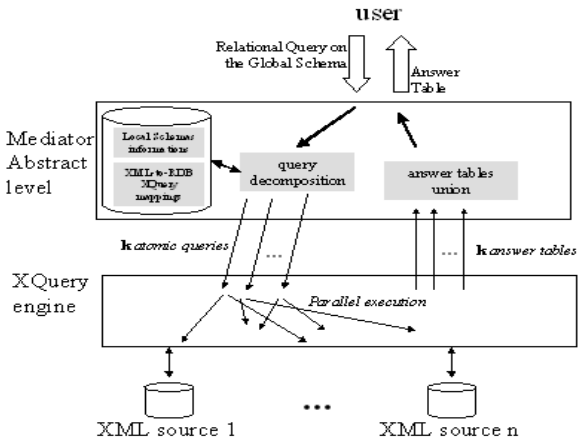
$$GE(RelA)JoinGE(RelB) =$$

$$\bigcup_{(1\leq i\leq m)} exec(XV(RelA,di))\ Join(\bigcup_{(1\leq j\leq m)} exec(XV(RelB,dj)))$$

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:

$$GE(RelA)JoinGE(RelB) = \bigcup_{i,j}(exec(XV(RelA,si)Joinexec(XV(RelB,sj))$$

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 10 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.

Fig. 7.

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 7.

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 9 and in Query 10). 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 9 lines 3-10, Query 10 lines 3-7) directly in the navigational context of a single optimized atomic-join-query (see Query 11). 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 11). 2) Choosing which attributes return in the RETURN clause (lines 19-22 in the Query 11).

```
<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>
```

Fig. 8. Query1 -atomic join query (exec (XV(RelA, si) Join exec(XV(RelA,sj))

3 Information integration by semantically enriching data tables

This second method focus on the data tables contained in the documents found on the Web. The method takes advantage of the presence of relational structures to automatically transform these data in order to make them compatible with the schema of the existing database .

3.1 Preliminary notions

We first present the generic XML representation of tables – called XTab. Then we introduce the ontology of the application domain. That section ends with a very preliminary example of what the result of the semantic enrichment is.

```
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>
```

Fig. 9. Query2-view XV(FoodproductMicroorganismTemperature , lab-data2004.xml)

```
1.  <TABLE>
2.  {
3.  let $TAB2-Root := doc("catalog.xml")/CATALOG
4.  for $TAB2-lev-AC in $TAB1/FOODPRODUCT
5.  for $TAB2-lev-ACE in $TAB1-lev-AC/TEST
6.    let $TAB2-FoodProduct-COLUMN := $TAB2-lev-AC/LABEL
7.    let $TAB2-Factor-COLUMN := $TAB2-lev-ACE/REPORT/FACTOR

8.  return
9.    <TUPLE>
10.   <FOODPRODUCT>{ $TAB1-FoodProduct-COLUMN /text()}</FOODPRODUCT>
11.   <FACTOR>{ $TAB1-Factor-COLUMN /string()}</FACTOR>
12.   </TUPLE>
13. }
14. </TABLE>
```

Fig. 10. Query3 - view XV(FoodproductFactor, catalog.xml)

The XTab format The data tables are first represented in XML, using purely syntactic tags that are domain-independent. The tables are automatically represented using a list of lines, each line being composed of a list of cells. Besides, when it is possible, titles are extracted. This format called XTab has been defined in the e.dot project [e.dot2004]. More complex structures of tables need heuristics such as [Pivk, Cimiano, & Sure2004] in order to be translated into

```
1.  <TABLE>
2.  {
    mapping for relation FoodproductMicroorganismTemperature on source lab-data2004.xml:
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()
    mapping for relation FoodproductFactor on source file catalog.xml
11. let $TAB2-Root := doc("catalog.xml")/CATALOG
12. for $TAB2-lev-AC in $TAB1/FOODPRODUCT
13. for $TAB2-lev-ACE in $TAB1-lev-AC/TEST
14.   let $TAB2-FoodProduct-COLUMN := $TAB2-lev-AC/LABEL
15.   let $TAB2-Factor-COLUMN := $TAB2-lev-ACE/REPORT/FACTOR

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

17. return
18. <TUPLE>
19. <FOODPRODUCT>{ $TAB1-FoodProduct-COLUMN }</FOOD_PRODUCT>
20. <MICROORGANISM>{ $TAB1-Microorganism-COLUMN }</MICROORGANISM>
21. <TEMPERATURE>{ $TAB1-Temperature-COLUMN }</TEMPERATURE>
22. <FACTOR>{ $TAB1-Factor-COLUMN /string()}</FACTOR>
23. </TUPLE>
24. }
25. </TABLE>
```

Fig. 11. Query4 - optimized atomic join query (XV(RelA,si) Join XV(RelA,sj))

this simple XTab structure. These heuristics are not presented here. The XTab representation of Figure 12 is shown in Figure 13.

Products	pH values
Cultivated mushroom	5.00
Crab	6.60

Fig. 12. *approximative pH of some food products*

The Sym'Previus ontology The Sym'Previus project [sym] has developed an ontology dedicated to the risk assessment domain. In order to exploit the data tables and query them through the MIEL++ system – which is based on the Sym'Previus ontology – we have to express data using the vocabulary stored in that ontology. The Sym'Previus ontology is composed of:

```
<?xml version="1.0" encoding="UTF-8"
standalone="no"?>
<table><title> <table-title>
approximative pH of some food products</table-title>
<column-title>Products</column-title>
<column-title>pH values</column-title></title>
<nb-col>2</nb-col>
<content>
<line>
<cell>cultivated mushroom</cell>
<cell>5.00</cell>
</line>
<line>
<cell>crab</cell>
<cell>6.60</cell>
</line>
</content></table>
```

Fig. 13. *XTab Representation of figure 1*

1. a term taxonomy which contains 428 terms of the domain (food, microorganism, experimental factors, ...) which are organized by the specialization relation $\preceq$;
2. a relational Reference schema that contains 25 semantic relations between terms of the taxonomy. A semantic relation r is characterized by its signature $attrs(r)$ composed of the set of attributes of the relation. The elements of $attrs(r)$ belong to the term taxonomy. For instance, the relation *foodFactorMicroorganism* has the signature *(food, factor, microorganism).*

3.2 Very preliminary example

Thus, we enrich XTab documents with tags and values provided by the ontology. More precisely, we have defined a representation formalism named SML – *Semantic Markup Language* – where table lines are not represented by cells anymore but by a set of semantic relations between columns.

Let us consider the semantic relation named *foodPH* which links a food product with its pH value in the ontology. The aim of the enrichment is to reformulate an XTab document such as Figure 13 in an SML document such as Figure 14. In this SML document, the semantic relation *foodPH* which has been recognized in the table is represented and instantiated using the table values.

In order to instantiate the relation, we try to associate one or several terms of the taxonomy with each value of the table. If the value does not appear directly in the taxonomy, we use mapping techniques in order to find similar terms. In the example, the first column value *crab* belongs to the taxonomy. But the value *cultivated mushroom* does not appear in the taxonomy; nevertheless, we propose to associate *mushroom* with it thanks to a mapping procedure. This value is represented in the SML tag $< ontoVal >$ while the original value is kept using the tag $< originalVal >$. Thus, the original value can be shown in the result of a query, even if the query is asked on a value belonging to the ontology. This

```
<table> <title><table-title>
approximative pH of some food products </table-title>
<column-title> Products </column-title>
<column-title>pH values</column-title>...
</title> <content>
<rowRel>
<foodPH>
<food><ontoVal>mushroom</ontoVal>
<originalVal> cultivated mushroom </originalVal>
</food>
<ph><ontoVal/>
<originalVal>5.00</originalVal></ph> </foodPH>
</rowRel>
<rowRel>
<foodPH>
<food><ontoVal>crab</ontoVal>
<originalVal>crab</originalVal></food>
<ph> <ontoVal/><originalVal>6.60</originalVal> </ph>
</foodPH> </rowRel>
</content> </table>
```

Fig. 14. *Simplified SML Representation of Figure 1*

SML representation conforms to the SML DTD (Document Type Definition) we have defined in [e.dot2004].

3.3 Identification of the columns of the data table

In order to extract the relations of the table, we perform two steps. The first one, presented in this section, consists in identifying a term of the taxonomy which represents each column of the data table. The second step, presented in the next section, will consist in discovering semantic relations between data table organized in columns.

The identification of the columns of the data table is based on two pieces of information: the content of the column which is mainly used, and the title of the column, which is used in case the content of the column is not helpful enough.

The content of the column is used as follows: we try to associate a term of the ontology taxonomy with each value belonging to the column. Then we search for common generalizers – "subsumers" of these terms. The use of a threshold allows us to associate a generalizer with a given column even if we have not recognized all the values of that column.

Definition 3. *An* A-term *is a term of the taxonomy that appears at least one time as an attribute of a relation signature in the relational reference schema of the ontology. The set of all A-terms is noted* AT.

We first try to find values of the columns that belong to the taxonomy or that are included[4] in one term of the taxonomy. We then look for an A-term which subsumes almost all the values in the term taxonomy. First, an A-term

[4] in the sense of the inclusion of sets of words, after a lemmatization step and without taking the "empty" words (determiners or prepositions) into account

can be associated with a column Col if and only if the rate of the subsumed values is greater than a given threshold th. The set of all A-terms that verify this constraint is noted $ATCandidate(Col, th)$:

$$ATCandidate(Col, th) = \{t \mid t \text{ in } AT \text{ and } \frac{|sub(t, Col)|}{|Col|} \geq th\}$$

where *sub(t,Col)* is the set of values of Col that are subsumed by the A-term t.

Among these candidates, we select the most specific A-terms that subsume the largest set of values. This set of representative A-terms is noted $ATRep$:

$$\begin{aligned} ATRep(Col, th) = \{t \mid t \in ATCandidate(Col, th), \\ \neg\exists\ t' \text{ such that } t' \in ATCandidate(Col, th) \\ \text{and } |sub(t', Col)| > |sub(t, Col)|, \\ \neg\exists\ t'' \text{ such that } t'' \in ATCandidate(Col, th) \\ \text{and } |sub(t'', Col)| = |sub(t, Col)| \text{ and } t'' \preceq t\} \end{aligned}$$

If there is more than one A-Term in ATRep, we keep the first one. In fact, experiments have shown that if the threshold is high enough there is zero or one representative A-term.

If no representative A-term has been found by using this procedure, we exploit the title of the column if it is available.
We exploit the values of the column first because if we are able to identify an important number of values, the A-term is often relevant. Besides, the treatment of the title can lead us to a misunderstanding association. If no A-Term has been found, we keep the column in the SML document and we associate the generic A-term named *attribute* with it.

Products	**Qty**	**Lipids**	**Calories**
whiting with lemon	100 g	7.8 g	92 kcal
ground crab	150 g	11.25 g	192 kcal
chicken	250 g	18.75 g	312 kcal

Fig. 15. *Nutritional Composition of some food products*

In the table of Figure 15, the terms *crab* and *chicken* belonging to the ontology have been associated with the values *ground crab* and *chicken*. If the threshold is 0.5, the most specific A-term that subsumes these two terms is the A-term *Food*. The second column has not been identified because it only contains numeric values and the title is an abbreviation; the generic A-Term *attribute* is associated with it. *lipid* and *calorie* have been associated with the last two columns thanks to the exploitation of their titles.

Definition The schema tabSch *of a table* tab*, noted* tabSch(tab)*, is the finite set of couples* $(col, ATRep(col, th))$ *that can be found for a given threshold* th*.*

$$tabSch(Tab) = \{(col, t) | t \in ATRep(col, th)$$

$$or\ [(t = attribute)\ and\ ATRep(col, th) = \emptyset]\}$$

The schema of the table *Tab2* shown in Figure 15 is:
tabSch(Tab2)={(1,food) (2,attribute) (3,lipid),(4,calorie)}

3.4 Identification of the semantic relations appearing in the data table

We present now how we identify one or several semantic relations in the schema of the table. That identification is done by comparing the "natures" of the columns identified during the previous step with the attributes appearing in the signatures of the semantic relations of the ontology of the domain. Of course, an exact mapping between the schema of the table and the signature of a specific semantic relation is the ideal case. In most of the cases, we will obtain several possible mapping with subsets of the attributes of the schema of the table. Or we will have only partial mapping, with only a subset of the attributes of the signature of a relation, etc. So we will see that we propose an automatic identification of the semantic relations as flexible as possible.

We say that a relation is *completely represented* if each attribute of its signature subsumes or is equal to a distinct A-term of the table schema.

Thus, suppose that the three relations *foodLipid*, *foodCalorie*, *foodPh* belong to the ontology and that the two relations *foodLipid* and *foodCalorie* mean "the number of lipid (or calories) contained in 100 g of the foodstuff", because the experts have considered that the weight is normalized. In table of Figure 15, the relations are extracted in the following way:

Definition 4. *foodLipid*, is completely represented by the values found in the first and the third columns.
- *foodCalorie* is completely represented by the values found in the first and the fourth columns.

Since the second column *qty* is not identified and does not participate to any of these two relations, we add to each relation a generic attribute which will contain values found in this second column. If this attribute was not represented, for example, the third line of the table would be interpreted as *"**100g** of chicken correspond to 312 calories"*. When the generic attribute is taken into account,

the interpretation is "***250g*** *of chicken correspond to 312 calories*". So, in such cases, the representation of additional information leads to better interpretations of the data.

Figure 16 proposes the SML representation of the relations *foodLipid* and *foodCalorie*.

```
<table>
<content>
<rowRel additionalAttr="yes">
<foodLipid relType ="completeRel">
<food>...</food> <lipid> ... </lipid>
<attribute> ...</attribute>
</foodLipid>
<foodCalorie relType ="completeRel">
<food>...</food> <calorie> ... </calorie>
<attribute> ...</attribute>
</foodCalorie>
</rowRel> ...
</content> </table>
```

Fig. 16. *SML representation of completely represented relations*

We say that a relation is *partially represented* if it is not completely represented and if at least two attributes of its signature subsume or are equal to different A-terms of the schema of the table. We have considered partially represented relations in order to take the following two cases into account.

Partially represented relations with Null attributes:

This is the case when an attribute of the semantic relation has not been associated to column of the table schema. For example in the table of Figure 15, the semantic relation *foodAmountLipid*, defined in the ontology on its attributes *food*, *amount* and *lipid*, is partially represented in the table schema *tabSch*, since the attribute *amount* is not represented in the table schema. Figure 17 presents the SML representation of *foodAmountLipid* relation :

Note that when a relation is partially represented, the attributes that do not appear in the schema are represented in the SML document by means of an empty tag like
<*amount* ***attrType="Null"***/>. In this example, the generic attribute represents precisely the missing attribute *Amount*.

Partially represented relations with constant values:

This is the case when one of the relation attributes correspond to a constant value which appears in the title of the table.

Let *tabSch* the table schema computed from the table tab_3 of Figure 18: $tabSch(tab_3) = \{(1,food),(2,factor)\}$.

In this table schema, the relation *foodFactorMicroorganism* is partially repre-

```
<table>

<content>
<rowRel additionalAttr="yes">
...
<foodAmountLipid relType ="partialNull">
<food attrType="Normal">...</food>
<amount attrType="Null"/ >
<lipid attrType ="Normal"> ... </lipid>
<attribute attrType="generic"> ...</attribute>
</foodLipid>

</rowRel> ...
</content> </table>
```

Fig. 17. *SML representation of a partially represented relation with Null attributes*

Products	Doubling time (h)
Minced meat	30[1]
Cured raw pork	3.6[1]
Frankfurters	9[1]

Fig. 18. *Doubling times of Listeria monocytogenes in foodstuffs*

sented: the attributes *food* and *factor* are represented in the table schema and the attribute *Microorganism* is represented by a constant value *Listeria Monocytogenes* which appears in the table title "Doubling time of ***Listeria Monocytogenes*** in foodstuffs".

This constant is used as a value for the corresponding attribute of the semantic relation and it is propagated into all the instances of the relation. Figure 19 presents the SML representation of the *foodFactorMicroorganism* relation.

```
<table>
<content>
<rowRel additionalAttr="no">
...
<foodFactorMicroorganism relType ="partialConst">
<food attrType="Normal">...</food>
<factor attrType="Normal"> ... </factor>
<microorganism attrType ="Const"> listeria monocytogenes
</microorganism>
</foodFactorMicroorganism>
...

</rowRel> ...
</content> </table>
```

Fig. 19. *SML representation of partially represented relations with attributes in constants*

Because we want to keep unidentified data, we also add to the semantic relations we have found the set of generic attributes of the table schema. This is done even if the relation is partial. Actually, one of these additional attributes may be a missing attribute of the relation. Besides, this attribute can add a contextual information which may modify the user's interpretation of the relation.

When no relation has been found in the table schema, a generic relation named *relation* is generated in the SML document. In this way, we keep semantic links between values even if this link has not been identified. Thus, it is possible to query the SML documents by means of lists of key-words.

3.5 Instantiation of the semantic relations

Once the relations are extracted, we instantiate them by the values contained in the table. Besides, terms of the ontology are associated with each value when it is possible. The SML formalism allows us to associate several terms that can be found by different mapping mechanisms. We have considered two kinds of mapping procedures.

The first one uses simple syntactic criteria. Each value is considered as a set of lemmatized words M_v where empty words such as determiners or prepositions are suppressed. The same treatment is applied to the terms of the ontology. Then, we consider that there may exist a semantic similarity between a value v and a term t if :

1. equality: ($M_v = M_t$)
2. inclusion: ($M_v \subset M_t$ or $M_t \subset M_v$)
3. intersection: ($M_t \subset M_v$) or ($M_v \cap M_t \neq \emptyset$).

These three criteria are applied using the previous order.

The second mapping procedure uses more semantic criteria. Actually, we have chosen to use the unsupervised approach PANKOW – Pattern-based Annotation trough Knowledge On the Web [Cimiano, Handschuh, & Staab2004] where patterns are used to categorize proper nouns (instances) with regard to an ontology. PANKOW applies a set of linguistic patterns including Hearst patterns [Hearst1992] (i.e. the $< concept > < instance >$, $< concept >$ such as $< instance >$, ...) on the biggest corpus available: the World Wide Web. In fact, they exploit the google API and take the number of pages in which patterns appear as an indicator for the strength of the pattern. We have used the same approach on data table even if they are not necessarily proper nouns. We have applied the general pattern "$< value >$ is a $< term >$" in order to discover specialization relations between values and terms of the ontology using the Web corpus. For a given value, we instantiate the pattern with each term of the domain ontology and keep the best term with regard to the number of pages. Because of the specificity of our domain, the number of pages can be very low. For instance, when we try to associate the value "ice cream" to a term of the

ontology, the pattern "ice cream is a dessert" is found in 35 pages. Happily, "ice cream is a microorganism" is not found. Note that the term *dessert* cannot be found by our syntactic criteria.

Figure 20 shows a part of the SML document which is automatically generated from the XTab document of Figure 15. This document is structured in the following way:

```
<table>
<table-title>Nutritional Composition of some food products
</table-title >
<column-title> Product </column-title>
<column-title>Qty</column-title>
<column-title>lipids</column-title>
<column-title>calories</column-title>
<column-nb> 4 </column-nb>
<content>

<rowRel additionalAttr="yes">
<foodLipid relType="completeRel">
<food indProc="yes" attrType="Normal">
<ontoVal indMap="intersection"> whiting Provencale
</ontoVal>
<ontoVal indMap="intersection"> green lemon </ontoVal>
<ontoVal> whiting fillets </ontoVal>
<originalVal> whiting with lemon </originalVal>
</food>
<lipid indProc="no" attrType="Normal">
<ontoVal indMap="notFound"/>
<originalVal> 7.8 g</originalVal>
</lipid>
<attribute indMap="notFound" indProc="no"
attrType="Generic">
<ontoVal/>
<originalVal> 100 g</originalVal></attribute>
</foodLipid>
<foodCalorie relType="completeRel"> ... </foodCalorie>
<foodAmountLipid relType="partialNull"> ...
</foodAmountLipid>
</rowRel>
...
</content> </table>
```

Fig. 20. *SML Representation of the nutritional composition of food products*

The main part of the document is inside the *content* element. It represents the table like a set of lines where each line is now a set of semantic relations (like, for example, ***foodLipide or foodCalories***).
The SML representation of a relation is composed of the set of attributes that appear in the signature of the relation described in the relational Reference Schema of the ontology (e.g. *foodLipid*(*food*, *lipid*)). Each attribute subsumes the representative term of the column or subsumes a term which has been found in its title. A set of terms represented inside the XML tag *ontoVal* is associated with each value. Thus, *crab* has been associated with *ground crab* while three different

terms are proposed for *whiting with lemon* : *whiting Provencale*, *green lemon* and *whiting fillets*. The original value is kept inside the XML tag *originalVal*.

The generality of the SML representation is ensured by the possibility of an automatic generation of the SML DTD from an ontology which contains a taxonomy and a relational reference schema.

3.6 Interrogation of SML documents

Some indicators that can be exploited in the queries Our approach allows one to extract data from tables even if we are not sure of their representation using the vocabulary of the ontology. It is the reason why we have defined a list of indicators that are represented in the SML document and that will be exploited during the query evaluation.

We present now the two main treatment indicators represented in SML as XML attributes attached to lines or to relation attributes. The first one is related to the structure of the relations (presence or absence of additional attributes).

additionalAttr: it informs on the presence of one or several additional attributes that represent the columns of the table which could not be associated with an identified relation. It is added to the tags $< rowRel >$ of SML document. For example in the table of Figure 15, this indicator allows the query engine to use the generic attribute associated with the *Quantity* column.

The following indicator make it possible to specify the kind of mapping procedure used to find a term of the ontology; it can thus be used to evaluate the risk of a mapping error. It is added to the $< ontoVal >$ tags of the SML document.

indMap: it indicates the name of the mapping procedure (inclusion, intersection or PANKOW) used to find the term of the ontology which corresponds to the original value of the table. Several mapping operators can exist in the application, this indicator allows us to modulate a trust degree, relating to enrichment, according to mapping operators. Besides, it can be used to visualize the original value if necessary.

These treatment indicators can be used by the query engine to adapt and find other answers for the user in cases of dissatisfaction.

An example of interrogation To query SML documents, XQuery queries have been written. They rely on the SML DTD. In the following, we describe a query example where the user looks for the quantity of lipid in 100 g of crab. The evaluation of this query consists in searching in the SML document for the subtrees – SML fragments – such that the parent node is *foodLipid* and such that there is an element *ontoVal* that contains the value "crab". The indicators *indMap* and *indProc* are used to check the validity of the semantic enrichment of

the data. As the indicator *additionalAttr* has the value "yes", the query engine displays the additional information *150g*. This example shows how the unidentified attributes that are kept in the SML representation can increase the accuracy of the user interpretation. Besides, the original value *ground crab* is displayed since *indProc* indicates that a treatment was carried out on the original value. The evaluation of this query performed on the document of Figure 20 is presented in Figure 21.

<table>
<title> Nutritional composition of some food products </title>
<food> **ground crab**</food> <lipid>**11.25 g**</lipid>
<validity>inclusion</validity>
<additionalattr>**150 g**</additionalattr>
<category> unknown</category> </table>

Fig. 21. *A possible structure of the query answer*

3.7 First results

We present in this section the results of the first experimentation of our method. The approach has only be tested on the risk assessment domain represented in the Sym'Previus ontology. In this evaluation, we show the capacity of our system to recognize relations of the ontology in the XTab tables . Our goal was to compare the results provided with our automatic method with a manual one done by an expert. We compared the results in terms of the well-known information retrieval measures Precision, Recall and F-Measure.

Test set Among two hundred real XTab tables collected from the Web, we have selected 33 tables. One table is selected in the test set if and only if we identify, among its columns at least one semantic relation attribute represented in the ontology.

Evaluation methodology In order to evaluate our approach, we have distinguished the results found for the three kinds of semantic relations: the Completely represented Relations (CR), the Partially represented Relations where all the missing attributes are identified by Constants in the table title (PRC) and the Partially represented Relations which contain at least one attribute which

[4] The XTab tables are the result of an automatic transformation applied on HTML and PDF documents found on the Web

is not identified – Null attributes – (PRN). Note that PRC relation can only found in the tables which are associated with a table title.

In first step we run our prototype on the real test set of XTab documents. In second step a domain expert checks the relevance of each semantic relation provided by our system.

To identify the semantic relations represented in the table represented in the XTab document, the expert has access to the whole information of the original – *HTML or Pdf* – document but he only considers information which are contained in the XTab document (ie. the table title and the table content). The expert considers that a semantic relation is *correct* if the relation is represented in the table and if all its attributes are correctly identified. If he recognizes in the XTab document one semantic relation which is not found by our system, he considers that the relation is *forgotten*. By this way, he can determine which semantic relations provided by our system are incorrect and which are forgotten.

In Figure 22, we show the result of this step for each kind of relation (CR, PRC and PRN) : number of semantic relations which have been found by the system, incorrect semantic relations and forgotten semantic relations.

	Found rels	**Incorrect rels**	**Forgotten rels**
CR	30	22	11
PRC	6	3	5
PRN	23	2	3

Fig. 22. *Expert results after semantic relations checking step*

On these results we have computed Recall, Precision and F-Measure. Let T, T' be two variables that represent the semantic relation type considered in the three measures calculations. It gets values in : {CR, (CR and PRN), (CR, PRN and PRC)}. Let *Correct_Rels(*T*)* be the number of semantic relations of type T, correctly found by our system.

$$Correct\_Rels(T) = Found\_Rels(T) \; - \; Incorrect\_Rels(T)$$

Recall is the percentage of relations (all types) actually represented in the data tables and correctly found by our system. Here we suppose that T'={*CR, PRN and PRC*}.

$$Recall = \frac{Correct\_Rels(T)}{(Correct\_Rels(T')) + \; Forgotten\_Rels(T'))}$$

Precision Is the percentage of relations found in the data tables by our system and with a correctly assigned relation signature.

$$Precision = \frac{Correct\_Rels(T)}{Found\_Rels(T)}$$

F-Measure as usual we balance Recall and Precision against each other.

$$F - Measure = \frac{2 * Recall * Precision}{Recall + Precision}$$

Results The diagram presented in Figure **??** gives the results in term of precision, recall and F-Measure of our semantic enrichment system approach. The first interesting observation is that the recall value increases significantly when our system takes into account partially represented relations. This result shows clearly the interest of the partially identified semantic relations kept in the SML documents, even when missing attributes are not identified by constants. If we restrict the relation types on the complete relations, we would have only 0.15 for the recall value, whereas in the case where we keep all the identified relations (ie. completely and partially represented relations) we have 0.62 for the recall value. Note that our aim is precisely to obtain a satisfying *Recall* value. Because we have chosen to keep all the identified pieces of information as well as information which are not completely identified such as partial relations, generic attributes end partial relations. We can also note that the precision is increasing as well. This result is globally shown by the increasing of the F-Measure value.

4 Conclusion

The two methods described in this paper for integrating and querying XML data 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 data extracted from the Web. 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 ways 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 documents coming from the Web by means of a relational-like query language is a point of interest of our two approaches.

The first approach proposes a way 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. To realize this, 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. Many researchers have studied the problem of storing XML documents into relational tables [**?**],[Amer-Yahia, Du, & Freire2004], [Florescu & Kossman1999], and also the converse problem of exporting relational data into XML [Halverson *et al.*2004], [Funderburk *et al.*2002]. 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.

The second approach proposes a way to integrate heterogeneous data tables by transforming these data in order to make them as much as possible compatible with the ontology (composed of a relational Reference schema and a taxonomy). The semantic enrichment is completely automatic and it is guided by an ontology of the domain. Thus, that processing cannot lead to a perfect and complete enrichment. The XML representation we propose keeps all the possible interpretation in order to let the possibility of using them during the query step, for example by allowing a query processing based on keywords or by exhibiting some relevant information to the user in order to help him/her during the interpretation of the results.

Then, in case of ambiguity, it is possible to associate several terms of the ontology or several semantic relations with a same set of columns. In order to allow the query processor to adapt its answers or to evaluate their relevance, we log the processes by means of a set of indicators.

The approach we propose is currently under testing in the domain of the food risk assessment, by means of a Java prototype. In order to query SML documents, we wrote *XQuery* queries which take advantage of the treatment indicators inserted in the SML documents. Those queries have been tested by means of the MIEL++ query engine.

Some works like [Kushmerick2000], [Muslea, Minton, & Knoblock2001] and [Hsu & Dung1998] allow to extract knowledge by learning rules from a sample of manually annotated documents. Our goal is quite different since our approach is completely automatic and exclusively guided by the ontology.

Moreover, the documents we use to fill the data warehouse are heterogeneous and, contrarily to previous approaches like [Crescenzi, Mecca, & Merialdo2002] and [Arasu & Garcia-Molina2003], we cannot base the search for information on a common structure discovered among a set of homogeneous documents.

The techniques we use to identify the columns of the table are based first on the values contained in those columns. [Rahm & Bernstein2001] and [Doan *et al.*2003] showed that those techniques give good results in the framework of the search for schema mappings for relational databases or XML. In our case, we do not have the schema of the tables we work on: we have to discover it first before searching for mappings with the semantic relations of the ontology.

We can now enhance our mapping operators, for example by using external resources such as WordNet or by using more sophisticated similarity measures [Robertson & Willett1998]. Moreover, we can think about using linguistic tools allowing to process the table content (cells, titles) represented in a more complex way.

Generality of both methods will be checked by applying them to another application domain.

References

[Amer-Yahia, Du, & Freire2004] Amer-Yahia, S.; Du, F.; and Freire, J. 2004. A comprehensive solution to the xml-to-relational mapping problem. In *WIDM '04: Proceedings of the 6th annual ACM international workshop on Web information and data management*, 31–38. New York, NY, USA: ACM Press.

[Arasu & Garcia-Molina2003] Arasu, A., and Garcia-Molina, H. 2003. Extracting structured data from web pages. In *Proceedings of the 2003 ACM SIGMOD international conference on Management of data*, 337–348. ACM Press.

[Bohannon *et al.*2002] Bohannon, P.; Freire, J.; Roy, P.; and Simeon, J. 2002. From XML schema to relations: A cost-based approach to XML storage. In *ICDE*.

[Buche *et al.*2004] Buche, P.; Dibie-Barthélemy, J.; Haemmerlé, O.; and Houhou, M. 2004. Towards flexible querying of xml imprecise data in a dataware house opened on the web. In *Flexible Query Answering Systems (FQAS)*. Springer Verlag.

[Chamberlin *et al.*2005] Chamberlin, D.; Florescu, D.; Robie, J.; Simeon, J.; and Stefanescu, M. 2005. Xquery: A query language for xml, w3c working draft.

[Cimiano, Handschuh, & Staab2004] Cimiano, P.; Handschuh, S.; and Staab, S. 2004. Towards the self-annotating web. In *WWW '04: Proceedings of the 13th international conference on World Wide Web*, 462–471. ACM Press.

[Clark & DeRose1999] Clark, J., and DeRose. 1999. Xml path language (xpath), version 1.0, w3c recommendation.

[Crescenzi, Mecca, & Merialdo2002] Crescenzi, V.; Mecca, G.; and Merialdo, P. 2002. Automatic web information extraction in the roadrunner system. In *Revised Papers from the HUMACS, DASWIS, ECOMO, and DAMA on ER 2001 Workshops*, 264–277. Springer-Verlag.

[Doan *et al.*2003] Doan, A.; Lu, Y.; Lee, Y.; and Han, J. 2003. Profile-based object matching for information integration. *Intelligent Systems, IEEE* 18(5):54– 59.

[e.dot2004] e.dot. 2004. Progress report of the e.dot project. http://www-rocq.inria.fr/gemo/edot.

[Florescu & Kossman1999] Florescu, D., and Kossman, D. 1999. Storing and querying xml data using an rdbms. *IEEE Data Engineering Bulletin* 22(3):27–34.

[Funderburk *et al.*2002] Funderburk, J. E.; Kiernan, G.; Shanmugasundaram, J.; Shekita, E.; and Wei, C. 2002. XTABLES: Bridging relational technology and XML. 41(4):616–??

[Halverson *et al.*2004] Halverson, A.; Josifovski, V.; Lohman, G. M.; Pirahesh, H.; and Mrschel, M. 2004. Rox: Relational over xml. In *VLDB 2004*, 264–275.

[Hearst1992] Hearst, M. A. 1992. Automatic acquisition of hyponyms from large text corpora. In *Proceedings of the 14th conference on Computational linguistics*, 539–545. Association for Computational Linguistics.

[Hsu & Dung1998] Hsu, C.-N., and Dung, M.-T. 1998. Generating finite-state transducers for semi-structured data extraction from the web. *Inf. Syst.* 23(9):521–538.

[Kushmerick2000] Kushmerick, N. 2000. Wrapper induction: efficiency and expressiveness. *Artif. Intell.* 118(1-2):15–68.

[Lee & Chu2001] Lee, D., and Chu, W. W. 2001. Constraints-preserving inlining algorithm for mapping xml dtd to relational schema. *Data Knowledge Engineering* 39(1):3–25.

[Muslea, Minton, & Knoblock2001] Muslea, I.; Minton, S.; and Knoblock, C. A. 2001. Hierarchical wrapper induction for semistructured information sources. *Autonomous Agents and Multi-Agent Systems* 4(1-2):93–114.

[Pivk, Cimiano, & Sure2004] Pivk, A.; Cimiano, P.; and Sure, Y. 2004. From tables to frames. In *International Semantic Web Conference*, 166–181.

[Rahm & Bernstein2001] Rahm, E., and Bernstein, P. A. 2001. A survey of approaches to automatic schema matching. *The VLDB Journal* 10(4):334–350.

[Robertson & Willett1998] Robertson, A., and Willett, P. 1998. Applications of n-grams in textual information systems. In *Journal of Documentation*, 48–69.

[sym] http://www.symprevius.net.