A Two Layered Approach for Querying Integrated XML Sources

Felipe Victolla Silveira and Carlos A. Heuser
UFRGS – Porto Alegre, Brazil
victolla,heuser@inf.ufrgs.br

Abstract

The problem of data integration (query decomposition, data fragmentation) has been widely studied in literature, but the inherent hierarchical nature of XML data presents problems that are specific to this data model. Each many-to-many conceptual relationship must be mapped to a specific hierarchical structure in XML. Different XML sources may implement the same many-to-many conceptual relationship in different ways. In our approach the problem of integration of XML data sources is decomposed in two problems: (1) that of fragmentation of a global graph-like model (e.g., an ER model) into several local graph-like models conceptually representing data sources and (2) that of mapping the local graph-like model into an XML tree-like schema. This paper presents a set of fragmentation operators specifically designed for our approach, as well as a query decomposition mechanism that allows a query stated at the conceptual level to be decomposed into an XQuery statement at the XML level. As the query language at the conceptual level, we adopt CXPath (conceptual XPath) a query language we have defined in previous work.

Keywords: *XML*, *data integration*, *data fragmentation*, *query decomposition*

1 Introduction

In this paper we handle the problem of querying several XML data sources that possibly have different schemata and relate to a common domain. We apply the mediator approach [24], in which queries are submitted against a mediated or global schema. In this context an important decision is to choose the abstraction level of the data model used at the global level. One approach taken by some authors is to use the same abstraction level at the global and local levels [19, 5, 11, 12]. In the integration of XML sources this means that the global schema is an XML model. This approach has the potential advantage of simplifying the translation of queries from the global schema to the local schemata, but presents also a major problem: due to the inherent hierarchical structure of XML documents, non-hierarchical conceptual constructs like many-to-many relationships must be hierarchically represented in the XML schemata.

In previous work we have followed another approach, namely that of using a more abstract data model at the global level. In [15, 16, 6] we propose the use of an entity-relationship like conceptual model for the global and local levels, and show how each local XML schemata is abstracted to a local conceptual schema, and how these local conceptual schemas are integrated in the global conceptual schema. Queries against the conceptual model are stated in CXPath (conceptual XPath), an XPath based language, and translated into regular XPath by a query rewriting approach [3].

However, the mechanism proposed in [3] is limited to the translation of one source at a time. The local queries generated by this mechanism are stated against one single source, limiting this approach to queries that don't need to interact with several sources to build the answer. In this work we address the problem of decomposing a global query stated against the conceptual model into local queries stated against several sources. Given this context, there are two major problems which need to be handled. One is to define how a global conceptual schema is fragmented into several local schemas. The other problem is to define an algorithm that decomposes a global query into several local queries and then constructs the answer.

In this paper, we propose a solution for both problems. First, fragmentation operators for XML data are introduced, considering the existence of a global conceptual schema. We define three fragmentation operators against the conceptual model and the conceptual base: *split* fragmentation, *vertical* fragmentation and *horizontal* fragmentation. Additionally, we propose a query decomposition algorithm, based on the use of mapping information between the conceptual model and the XML documents. This algorithm generates XPath expressions to access the individual sources, and then integrates this queries into a single XQuery [23] expression, handling the fragmentation and constructing the answer for the query. The behavior of this algorithm is based on how the sources are fragmented. It means that it implements fragmentation reduction algorithms, similarly to what is done in distributed databases [18].

This paper is organized as follows. Section 2 presents the related work and better motivates our approach. Section 3 presents the data model and the query language used in our approach. Section 4 describes the fragmentation operators. Section 5 describes the mapping information and Section 6 describes the decomposition algorithm and its application to an example. Section 7 is dedicated to the conclusions and future work.

2 Related Work

The problem of fragmentation of XML documents has been investigated by several authors [11, 12, 2, 1]. Most of these work follow the approach of generalizing relational fragmentation operators to the XML data model [11, 12, 2]. Schewe et.al. [11, 12] present three fragmentation operators: vertical fragmentation, horizontal fragmentation and the split operator. The vertical and horizontal fragmentation operators are generalizations of the relational operators, while the split operator is based on a previous work of the same group dealing with fragmentation in the object-oriented model [21].

However, the hierarchical and navigational nature of XML introduces additional issues into the problem of fragmentation. As XML documents are trees, non-hierarchical conceptual constructs like many-to-many relationships may be represented in XML in several different ways (corresponding to different traversal directions of a relationship). Further, a relationship at the conceptual level may be represented in two different ways in XML. One solution is to employ an explicit parent-child relation (the *navigational* solution) and the other is to implement the relationship by associations between key values, as in the relational model (the *associative* solution).

Thus, defining fragmentation operators directly over the XML data model means that one has to deal in a single step with what actually are two different problems: that of abstracting from representation details in XML and that of dealing with fragmentation of data in several sources itself. In our approach we clearly separate both problems. Although aiming at XML fragmentation, our fragmentation operators are defined at the conceptual level, i.e., they define how a global mediated conceptual schema is fragmented to local conceptual schemata. The problem of abstracting from XML representation details is handled by the mapping from each local XML schema to the corresponding local conceptual model. Given the context of fragmentation in XML documents, we are unaware of any work in literature which follows this approach.

There is vast literature dealing with the problem of query processing over several and heterogeneous sources . Most of these work adopt the mediator approach [20, 14, 13, 10]. In the Agora Integration System [14], the global model is XML, and the integrated sources can be relational or XML. Agora adopts the LAV (local-as-view) approach, in which local sources are defined as views against the global schema. The Yacob mediator [20] adopts a different approach. It uses a graph-like global model, based on RDF, that defines concepts and relationship between concepts, being similar to the work presented here in many aspects. However in [20] the author focus on the construction of a query execution engine defining its architecture and functionalities, whereas in our approach we focus on the fragmentation operators and on the query decomposition algorithm. In this sense, our work complements the work done on the Yacob system [20]. To the best of our knowledge, there is no work in literature concerning query processing in XML that is based on the fragmentation of the sources, despite the existence of several work that addresses fragmentation in XML.

3 Data Model and Query Language

This section presents the data model used at the conceptual level, as well as the query language CXPath, used to state queries at the conceptual level.

3.1 Conceptual Model

The conceptual model is a high level abstraction of the XML data source models. It is build by a bottom-up approach described in [15, 16]. First, each local XML schema is abstracted into a local conceptual schema and after that local conceptual schemas are integrated resulting in a global conceptual schema.

The conceptual model is a simplified version of the ORM model [8]. It defines *concepts* and *relationships* between concepts. There are two kinds of concepts: *lexical* and *non-lexical*. Lexical concepts represent objects that have a textual content. Lexical concepts abstract atomic XML elements, like #PCDATA elements or attribute values. Non-lexical concepts do not have a direct textual representation and are abstractions of XML elements that contain other elements. Relationships at the conceptual model level are abstractions of two kinds of constructs at the XML levels. A relationship may have a *navigational* implementation, i.e. it may represent an access path relationship at the XML level. A relationship may further have an *associative* implementation, i.e, it may represent an identifier reference between data in two different XML data sources. For details on the process of abstraction of an XML schemata into a conceptual schema, please refer to [16]. A non-lexical concept may

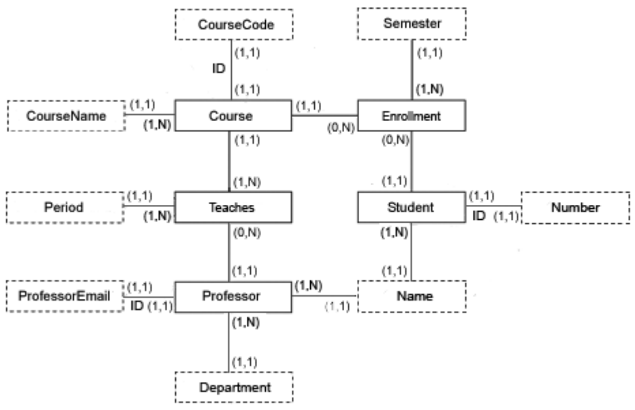


Figure 1. A conceptual schema

have an *identifier*. An identifier is a set of relationships that uniquely identifies an instance of the concept.

Figure 1 shows a conceptual schema for student enrollment in a university. There are five non-lexical concepts: *Course*, *Enrollment*, *Student*, *Teaches* and *Professor*, which are represented by solid rectangles. Each of these concepts is related to at least one lexical concept (dotted rectangle). For example, *Student* is related to *Name* and *Number*. The diagram represents also the cardinality of the relationships between concepts. For example, in this model, *Course* and *Enrollment* have a one-to-many relationship. The identifier relationships are labeled with 'ID' (for example *Student* is identified by the relationship with *Number*).

A formal definition of the conceptual model is given in [22].

3.2 CXPath

CXPath [3] is a query language, based on XPath [23], which is used to state queries at the conceptual level. Queries written in CXPath are *path expressions* over the concepts of a global schema. Path expressions may be enriched with *selection predicates*. Some examples of CXPath queries over the conceptual schema of figure 1 are presented below (for details please refer to [3].

Example 3.1. Retrieve the name of the students registered in the course with code "INF001" in the semester "2006/1".

```
/Course[CourseCode="INF001"]
/Enrollment[Semester="2006/1"]
/Student/Name
```

Example 3.2 Retrieve the name of the professors who will be teaching the course which code is "CSE 241".

```
/Course[CourseCode="CSE
241"]/Teaches/Professor/Name
```

4 Fragmentation Operators

In this work a set of fragmentation operations over a graph-like conceptual model is proposed. These operations have been originally developed for different data models. Horizontal and vertical fragmentations were defined over the relational data model [17, 4]. The split fragmentation was originally developed for the object-oriented data model [21] and afterward was adopted to the XML data model [11, 12]. The result of the fragmentation process is a set of local conceptual schemas and local conceptual bases, which are then directly mapped to XML documents by the process described in [16].

Notice that in our approach we clearly separate two different problems, the problem of fragmentation (decomposition of the global model into several local models) and the problem of abstraction of implementation details in an XML source into a local conceptual schema. If we apply fragmentation directly at the XML model level, we will have to deal with the problem that the same conceptual information may have several different representations at the XML level. In our approach, XML sources that have different schema but have conceptually the same content will be described by a single local conceptual model.

Below, we define first the local models that are obtained by the fragmentation operators and after that we discuss the fragmentation operators themselves.

4.1 Local Conceptual Model

A local conceptual model is a subset of a given global conceptual model. A global conceptual schema can be fragmented into several local schemas through fragmentation operators. The operators that produce local conceptual schemas are the vertical fragmentation and the split fragmentation, which will be detailed below.

When two non-lexical concepts that are associated by a relationship in the global conceptual model are fragmented into different local schemas, *identifier references* need to be included in these local schemas. An identifier reference is an information that implements a relationship between two local conceptual schemas in the same way a foreign-key implements a relationship between two relations in a relational database. The local schema in which an identifier reference is included will depend on the cardinality of the relationships. Each identifier reference is mapped to the identifier of the related non-lexical concept.

Figure 2 shows local schemas that may obtained by fragmentation of the conceptual schema on Figure 1. The non-lexical concepts *Enrollment*, *Course*, *Professor* and

Teaches, which are related in the global conceptual schema shown in Figure 1, were fragmented into the local conceptual schemas in Figure 2. Some relationships in the global schema are implemented in the local schemas through identifier references. For example, take the relationship between *Enrollment* and *Course* that appears at the global level. As the concepts *Enrollment* and *Course* are distributed in two different local schemas (LCM_3 and LCM_2 respectively) the relationship at the global conceptual level must be implemented through an identifier reference at the local level. In this example, the concept *CourseCode* was added to the local schema LCM_3. The instances of this concept are references the *Course* identifier in local schema LCM_2. The same idea is valid for the concept *ProfessorEmail* in LCM_2.

A formal definition of the local conceptual model is given in [22].

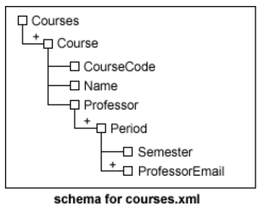
4.2 Operators

The *split fragmentation* operator is based on the operator presented in [11], where a given element of an XML document is replaced by a reference to a new element. This operator originates from previous work on object-oriented databases [21]. There, a complex operation in a class is replaced by a reference to a new class. In this work, the split operator has a different behavior. It takes a conceptual schema and divides it into two distinct local conceptual schemas. The relationships in the global schema that are splitted into different local schemas are implemented through identifier references. Figure 2 presents an example of the result of split operations applied to the conceptual schema in figure 1.

Vertical fragmentation is based on the corresponding relational operator [17], which produces fragments projecting subsets of attributes. In the context of this work, each fragment contains subsets of lexical concepts of the conceptual schema. This operator was also presented in [11, 12], but there it is defined over the XML data model, instead of the conceptual model. However, the behavior of both operators are very similar.

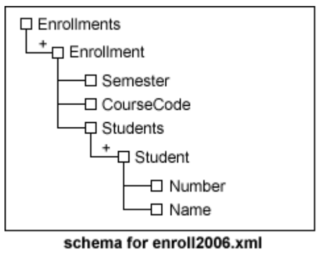
Horizontal fragmentation is also based on the corresponding relational operator [4], which produces fragments that correspond to subsets of the tuples. In the context of this work, this operator produces fragments that correspond to a subset of the instances of a given conceptual base. Each fragment is defined through a selection predicate. If the instance obeys the predicate, it is included in the fragment.

For more details and the formal definitions of this operators, please refer to [22].

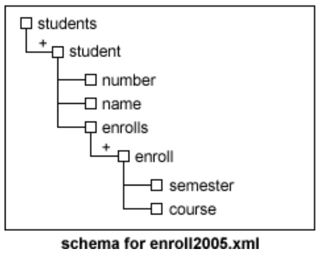


```
<Courses>
 <Course>
  <CourseCode>INF001</CourseCode>
  <Name>Database Foundations</Name>
  <Professor>
   <Period>
    <Semester>2001/6</Semester>
    <ProfessorEmail>prof1@inf.ufrgs.br</ProfessorEmail>
    <ProfessorEmail>prof2@inf.ufrgs.br</ProfessorEmail>
   </Period>
  </Professor>
 </Course>
</Courses>
```

Figure 3. Schemata for LCM_2 XML source



```
<Enrollments>
 <Enrollment>
  <Semester>2006/1</Semester>
  <CourseCode>INF001</CourseCode>
  <Students>
   <Student>
    <Number>001</Number>
    <Name>John</Name>
   </Student>
   <Student>
    <Number>002</Number>
    <Name>Debbie</Name>
   </Student>
  </Students>
 </Enrollment>
</Enrollments>
```



```
<students>
 <student>
  <code>008</code>
  <studentName>Eric</studentName>
  <enrolls>
   <enroll>
    <semester>2005/1</semester>
    <course>INF001</course>
   </enroll>
   <enroll>
    <semester>2005/1</semester>
    <course>INF002</course>
   </enroll>
  </enrolls>
 </student>
</students>
```

Figure 4. Schematas for LCM_3 XML sources

5 Mapping Between Local Model and XML Source

This section describes the mapping between a local schema and the corresponding local XML schemata. This mapping is used by the decomposition algorithm during the translation of a global query into a local query. We apply the global-as-view (GAV) approach [7]. The mapping approach presented here is an extension of the mapping approach we have previously defined for translating queries over a single data source [3].

Two types of mappings are applied: *absolute* and *relative* mappings. Absolute mappings are used to describe how a concept at the conceptual level is found at the XML level. Absolute mappings are given by absolute XPath expressions, one for each concept and each source. Relative mappings are used to describe how a relationship traversal at the conceptual level is mapped to a navigation between elements at the XML level. Relative mappings are given by relative XPath expressions that navigate from an element to another at the XML level.

Below we present examples of mapping information for the local conceptual schemas shown at figure 2, considering the structure of the XML files shown at figures 3 and 4.

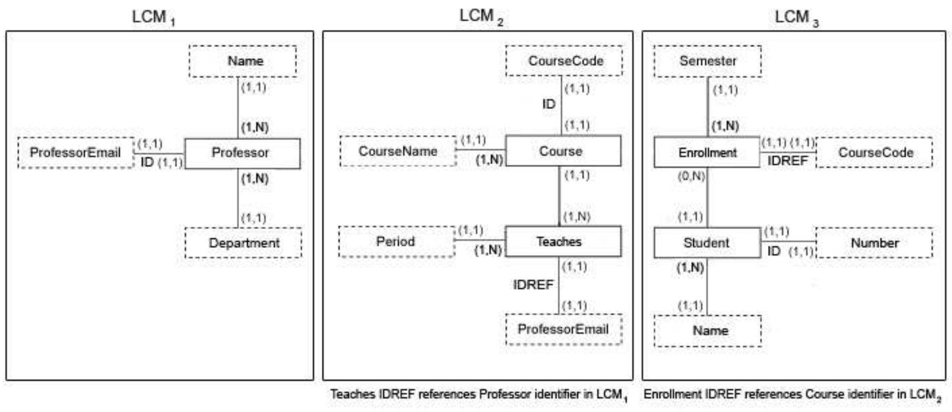


Figure 2. Local conceptual schemas produced through split fragmentation operations

Table 1. Absolute Mapping Information

concept	*LCM*	*url*	*expression*
Course	LCM_2	`courses.xml`	`/Courses/Course`
Teaches	LCM_2	`courses.xml`	`/Courses/Course /Professor/Period`
Enrollment	LCM_3	`enroll2005.xml`	`/students/students /enrolls/enroll`
Enrollment	LCM_3	`enroll2006.xml`	`/Enrollments /Enrollment`
Student	LCM_3	`enroll2005.xml`	`/students/student`
Student	LCM_3	`enroll2006.xml`	`/Enrollments /Enrollment/Students /Student`

Table 1 shows examples of absolute mappings. For each non-lexical concept in a given local conceptual schema and for each XML source, there will be one XPath expression that retrieves the XML elements which represent that concept. For example, consider LCM_2 (Figure 2) and the single XML source (*courses.xml*) that is associated to it (Figure 3). LCM_2 contains two non-lexical concepts: *Course* and *Teaches*. The concept *Course* at the conceptual level is mapped to the XPath expression `/Courses/Course` at the XML source. In the same way the mapping of the concept *Teaches* to this source is given by the XPath expression `/Courses/Course/Professor/Period`.

Table 2 shows examples of relative mappings. For each relationship traversal in a local conceptual schema and for each XML source, there is one relative XPath expression that defines how the relationship traversal at the conceptual level is mapped at the XML level. For example, consider the relationship between the concepts *Course* and *CourseCode* in the local conceptual schema LCM_2 (Figure 2). The traversal of this relationship from *Course* to *CourseCode* is mapped to the relative XPath expression `CourseCode`. This expression specifies that, in order to navigate in this XML source from the element that corresponds to the concept *Course* to the element that corresponds to the concept *CourseCode*, the relative XPath expression `CourseCode` must be applied. The relative XPath expression that maps the traversal of the same relationship in the reverse direction (from *CourseCode* to *Course*) is `..` (the XPath ancestor axis operator).

For a formal definition of the mapping information, please refer to [22].

6 Decomposition Algorithm

In this section, we describe how a query expressed at the conceptual level through a CXPath expression (Section 3.2) is rewritten into an XQuery expression at the XML level. More specifically, the decomposition algorithm first translates the CXPath expression into several XPath expressions, one for each XML source, and then groups these XPath subqueries into a single XQuery expression.

The main steps of the decomposition algorithm are the following:

1. **Parse the CXPath query**
2. **Handle horizontal fragments**
3. **Navigate inside fragments**
4. **Join split fragments**
5. **Implement the selection predicates**

The decomposition algorithm is explained here by discussing the results of each step of the algorithm on an example CXPath query. For details please refer to [22]. Further, due to lack of space the version of the algorithm explained here does not handle vertical fragmentation. At [22] the necessary changes in the algorithm to include this type of fragmentation are discussed.

In the following subsections we will show how the CXPath expression below (Example 3.1 of Section 3) is translated when it is executed against the global conceptual

Table 2. Relative Mapping Information

source	*destination*	*LCM*	*url*	*expression*
Course	CourseCode	LCM_2	`courses.xml`	`CourseCode`
Course	CourseName	LCM_2	`courses.xml`	`Name`
CourseCode	Course	LCM_2	`courses.xml`	`..`
CourseName	Course	LCM_2	`courses.xml`	`..`
Enrollment	Semester	LCM_3	`enroll2005.xml`	`semester`
Enrollment	CourseCode	LCM_3	`enroll2005.xml`	`course`
Enrollment	Student	LCM_3	`enroll2005.xml`	`../..`
Semester	Enrollment	LCM_3	`enroll2005.xml`	`..`
CourseCode	Enrollment	LCM_3	`enroll2005.xml`	`..`
Student	Enrollment	LCM_3	`enroll2005.xml`	`enroll/enrolls`
Student	Number	LCM_3	`enroll2005.xml`	`number`
Student	Name	LCM_3	`enroll2005.xml`	`name`
Number	Student	LCM_3	`enroll2005.xml`	`..`
Name	Student	LCM_3	`enroll2005.xml`	`..`
Enrollment	Semester	LCM_3	`enroll2006.xml`	`Semester`
Enrollment	CourseCode	LCM_3	`enroll2006.xml`	`CourseCode`
Enrollment	Student	LCM_3	`enroll2006.xml`	`Students/Student`
Semester	Enrollment	LCM_3	`enroll2006.xml`	`..`
CourseCode	Enrollment	LCM_3	`enroll2006.xml`	`..`
Student	Enrollment	LCM_3	`enroll2006.xml`	`../..`
Student	Number	LCM_3	`enroll2006.xml`	`Number`
Student	Name	LCM_3	`enroll2006.xml`	`Name`
Number	Student	LCM_3	`enroll2006.xml`	`..`
Name	Student	LCM_3	`enroll2006.xml`	`..`

Table 3. Result of *parseQuery*

oid	*concept*	*LCM*	*sources*	*predicate*
1	Course	LCM_2	`courses.xml`	`CourseCode= "INF001"`
2	Enrollment	LCM_3	`enroll2005.xml` `enroll2006.xml`	`Semester= "2006/1"`
3	Student	LCM_3	`enroll2005.xml` `enroll2006.xml`	
4	Name	LCM_3	`enroll2005.xml` `enroll2006.xml`	

schema described in Figure 1 and the local conceptual schemas described in Figure 2, with the mapping information declared in tables 1 and 2 being considered.

```
/Course[CourseCode="INF001"]
/Enrollment[Semester="2006/1"]
/Student/Name
```

6.1 Parse Query

The first decomposition step is to parse the query and generate a tuple for each concept in the CXPath query. CXPath subqueries that appear in the selection predicates are handled by recursive calls of this algorithm as explained below. Each tuple contains a tuple identifier (*object id*), the *concept*, the local conceptual model which the concept belongs to, its sources, and the selection predicates associated to the concept in the XPath expression.

The tuple identifier is used during the generation of the FOR clause of the resulting XQuery expression to name each variable with a $v followed by the object id. Table 3 contains the result of this step for the example query.

Table 4. Result of *handleHorizontalFragments* - FOR clauses

var	*expression*
`$v1`	`doc("courses.xml")/Courses/Course`
`$v2`	`doc("enroll2005.xml")/students/student /enrolls/enroll | doc("enroll2006.xml") /Enrollments/Enrollment`

6.2 Handle Horizontal Fragments

In this step, the algorithm begins to build the XQuery expression. For each local conceptual schema that is referenced by the concepts in the CXPath expression, one binding of an XQuery variable will be constructed. These bindings will be used in the FOR clause of the resulting XQuery expression. For the example query, the bindings in Table 4 will be constructed.

These variable bindings are constructed as follows. For each local conceptual schema, the algorithm takes the first concept in the parsed query (Table 3). In the example, these will be the concepts *Course* in LCM_2 and *Enrollment* in LCM_3. For each concept one binding is constructed. This binding contains a reference to each XML source that is associated to the local schema, as well as the absolute mapping expressions (Section 5) for the concept in each source, that specifies how instances of this concept are found in the XML source. For a definition of this algorithm, please refer to [22].

The first concept of LCM_2 found in the example query is *Course*. LCM_2 corresponds to the XML source `courses.xml` and the absolute mapping expression for

Table 5. Result of *navigateInsideFragments* - FOR clauses

var	*expression*
`$v3`	`$v2[base_uri(.)="enroll2005.xml"]/../.. | $v2[base_uri(.)="enroll2006.xml"]/Students/Student`
`$v4`	`$v3[base_uri(.)="enroll2005.xml"]/name | $v3[base_uri(.)="enroll2006.xml"]/Name`

Course in `courses.xml` is `/Courses/Course`. This leads to the first binding shown in Table 4. Local conceptual schema LCM_3 is represented by two different XML data sources, `enroll2005.xml` and `enroll2006.xml`. In this case, the binding will correspond to the union of the elements that represent the concept *Enrollment* in these sources (second line in Table 4).

6.3 Navigate Inside Fragments

In the second step we have defined variables that are bound to absolute XPath expressions that will be used to iterate over each fragment, i.e., over each local conceptual schema. In this step we will implement the navigation inside each fragment through relative XPath expressions. To handle this problem, we adopted the same approach as in [3].

The algorithm compares each pair of adjacent concepts in the parsed query. If the pair belongs to the same local conceptual schema, it means that a relationship between two concepts is being traversed. To implement this relationship traversal at the XML source level, the relative path expression defined by the relative mapping for this traversal (Section 5) is bound to a variable. The complete algorithm for this step is defined in [22]. For example, consider the relationship traversal from *Enrollment* to *Student* in LCM_3. This traversal is implemented in source `enroll2005.xml` by the relative XPath expression `../..`, and in source `enroll2006.xml` by the relative XPath expression `Students/Student` (see Table 2). This leads to the first line in Table 5.

These lines defines a binding for a new variable `$v3` that is bound to an expression that is relative to the `$v2` variable. Recall that $v2 is bound to an absolute XPath expression that retrieves the XML elements that represent the *Enrollment* concept, source of the relationship traversal being handled. The `$v3` binding constructed in this step specifies the traversal of the relationship from *Enrollment* to *Student* at the conceptual level. As this relationship is implemented in two different XML sources, the terms in form `base_uri(.)` are used to ensure that the relative path is being applied over the correct source.

Analogously a variable `$v4` is bound to the relative XPath expression that specifies the traversal from concept *Student* to concept *Name*.

Table 6. Result of *joinSplitFragments* - WHERE clauses

expression
`($v2[base_uri(.)="enroll2005.xml"]/course | $v2[base_uri(.)="enroll2006.xml"]/CourseCode) = $v1/CourseCode`

6.4 Join Split Fragments

A CXPath query may access concepts from different local schemas, which means that several split fragments (local schemas) need to be accessed and joined to answer the query. For each local schema, a variable bound to an absolute XPath expression that retrieves elements in the fragment was constructed in the second step. In this step, the criteria to join the elements from different sources is constructed. This criteria will take part of the WHERE clause in the resulting XQuery expression. The join criteria will be constructed using the *identifiers* and *identifier references* that have been defined during the the split operation, which generated the local schemas from the global schema (Section 3.1)

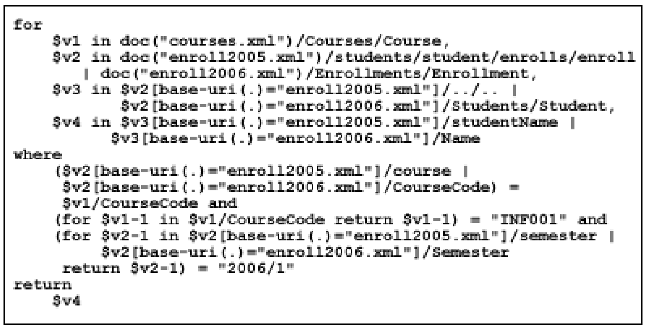
In the parsed query table (Table 3), for each adjacent pair of concepts that belong to different local schemas, a join must be defined. For each such pair of concepts, the algorithm finds the identifiers and the identifier references that are used to navigate from one fragment to the other. In the example (Table 3), there is a single pair of concepts that corresponds to the navigation from one fragment to the other, namely the traversal from concept *Course* to concept *Enrollment*. This relationship is implemented by identifier reference *Enrollment.CourseCode* that references the identifier *Course.CourseCode*.

Next, for each XML source, the algorithm takes the relative path expressions that implement the traversal from *Course* to the identifier *CourseCode* and from *Enrollment* to the identifier reference *CourseCode*. Finally, an equality predicate among this two relative XPath expressions is included in the WHERE clause, as shown in Table 6.

The algorithm for this step is defined in [22].

6.5 Handle Selection Predicates

This step of the algorithm is in charge of handling the selection predicates that may appear in the CXPath query. This selection predicates will be rewritten into terms in the WHERE clause of the resulting XQuery expression. The algorithm supports a simplified form of the selection predicate, which has the following structure:



```
for
    $v1 in doc("courses.xml")/Courses/Course,
    $v2 in doc("enroll2005.xml")/students/student/enrolls/enroll
        | doc("enroll2006.xml")/Enrollments/Enrollment,
    $v3 in $v2[base-uri(.)="enroll2005.xml"]/../.. |
           $v2[base-uri(.)="enroll2006.xml"]/Students/Student,
    $v4 in $v3[base-uri(.)="enroll2005.xml"]/studentName |
          $v3[base-uri(.)="enroll2006.xml"]/Name
where
    ($v2[base-uri(.)="enroll2005.xml"]/course |
     $v2[base-uri(.)="enroll2006.xml"]/CourseCode) =
     $v1/CourseCode and
    (for $v1-1 in $v1/CourseCode return $v1-1) = "INF001" and
    (for $v2-1 in $v2[base-uri(.)="enroll2005.xml"]/semester |
         $v2[base-uri(.)="enroll2006.xml"]/Semester
     return $v2-1) = "2006/1"
return
    $v4
```

Figure 5. XQuery generated by the decomposition algorithm to example 3.1

```
CXPathLeft comp [CXPathRight OR
constant]
```

where *CXPathLeft* and *CXPathRight* are CXPath subqueries, which may be relative or absolute expressions, *comp* $\in \{=, \neq, <, >, \leq, \geq\}$ is a comparison operator, and *constant* is a string.

The CXPath subqueries that may appear in the selection predicate are translated by a recursive execution of the decomposition algorithm.

For example, the selection predicate `CourseCode="INF001"` that contains an expression that is relative to the *Course* concept is translated into the predicate:

```
(for $v1-1 in $v1/CourseCode
 return $v1-1)
 = "INF001"
```

The XQuery expression `for $v1-1 ...` is obtained by the recursive execution of the decomposition algorithm on the relative CXPath expression `CourseCode` that appear in the CXPath selection predicate. When the CXPath expressions inside a selection predicate are relative, the context, i.e. the tuple identifier and the concept, must be passed as parameter to the decomposition algorithm. In the example, the parameters are the object id 1 and the concept Course. The details regarding to the recursive call are shown in [22].

The final XQuery for the example 2.1 is shown at figure 5.

7 Conclusion and Future Work

This paper presents an approach to handle the problem of querying integrated XML documents. We first present a set of fragmentation operators for XML documents that belong to a specific domain described by a conceptual schema, and then, on the basis of these fragmentation operators, present a query decomposition mechanism that translates a CXPath query expression at the global level into an XQuery expression at the XML level.

Our contribution is twofold. One contribution is to clearly separate the problem of fragmentation of a global schema into several local schemata from the problem of abstracting from different XML representations of the same conceptual information. This separation is achieved by the introduction of a conceptual layer at the local source level and at the global (mediated) level as well. The fragmentation operators are defined at this conceptual level. This concepts are formally defined in a separate document [22].

Another contribution is the query decomposition algorithm itself, which is based on the fragmentation of sources. The presented version of the algorithm handles split fragmentation and most of the horizontal fragmentation scenarios that can be described through the fragmentation operators. With the changes presented in [22], it can handle also the vertical fragmentation. A scenario that is not handled by this version of the decomposition algorithm is when the horizontal fragmentation operations are not disjoint - i.e. when the same instance appears in more then one XML source. We leave this issue for future work.

A further problem that is not handled by this version of the decomposition algorithm is query optimization. The queries generated by this algorithm are not the most efficient ones, and the performance of the query execution can be improved by rewriting the generated query. Future work should decide among alternatives like improving the generated XQuery or using a XML algebra (e.g., TAX [9]) after the decomposition is concluded.

Acknowledgments

This work is partially supported by CNPq (projects 473310/2004-0, and Gerindo) and FAPERGS (project).

References

[1] S. Bose, L. Fegaras, D. Levine, and V. Chaluvadi. A query algebra for fragmented xml stream data. In *DBPL*, pages 195–215, 2003.

[2] J.-M. Bremer and M. Gertz. On distributing xml repositories. In *WebDB*, pages 73–78, 2003.

[3] S. Camillo, R. Mello, and C. Heuser. Querying heterogeneous xml sources through a conceptual schema. *ER International Conference on Conceptual Modeling*, 2003.

[4] S. Ceri, M. Negri, and G. Pelagatti. Horizontal data partitioning in database design. In *SIGMOD '82: Proceedings of the 1982 ACM SIGMOD international conference on Management of data*, pages 128–136, New York, NY, USA, 1982. ACM Press.

[5] A. Doan, P. Domingos, and A. Y. Halevy. Reconciling schemas of disparate data sources: A machine-learning approach. In *SIGMOD Conference*, 2001.

[6] R. dos Santos Mello and C. A. Heuser. Binxs: A process for integration of xml schemata. In *CAiSE*, pages 151–166, 2005.

[7] A. Elmagarmid, M. Rusinkiewicz, and A. Sheth, editors. *Management of heterogeneous and autonomous database systems*. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 1999.

[8] T. Halphin. *Object-Role Modeling (ORM/NIAM). Handbook on Architectures of Information Systems*. Springer-Verlag, 1998.

[9] H. V. Jagadish, L. V. S. Lakshmanan, D. Srivastava, and K. Thompson. Tax: A tree algebra for xml. In *Revised Papers from the 8th International Workshop on Database Programming Languages*, pages 149–164. Springer, 2002.

[10] V. Josifovski and T. Risch. Query decomposition for a distributed object-oriented mediator system. *Distributed and Parallel Databases*, 11(3):307–336, 2002.

[11] H. Ma and K.-D. Schewe. Fragmentation of xml documents. In *SBBD*, pages 200–214, 2003.

[12] H. Ma, K.-D. Schewe, S. Hartmann, and M. Kirchberg. Distribution design for xml documents. In *ICeCE*, 2003.

[13] L. M. Mackinnon, D. H. Marwick, and M. H. Williams. A model for query decomposition and answer construction in heterogeneous distributed database systems. *J. Intell. Inf. Syst.*, 11(1):69–87, 1998.

[14] I. Manolescu, D. Florescu, and D. Kossmann. Answering xml queries on heterogeneous data sources. In *Proceedings of the 27th International Conference on Very Large Data Bases*, pages 241–250. Morgan Kaufmann Publishers Inc., 2001.

[15] R. Mello, S. Castano, and C. Heuser. A method for the unification of xml schemata. *Information and Software Technology*, 44:241–249, 2002.

[16] R. Mello and C. Heuser. A rule-based convertion of a dtd to a conceptual schema. *ER International Conference on Conceptual Modeling*, pages 133–148, 2001.

[17] S. Navathe, S. Ceri, G. Wiederhold, and J. Dou. Vertical partitioning algorithms for database design. *ACM Trans. Database Syst.*, 9(4):680–710, 1984.

[18] M. T. Ozsu and P. Valduriez. *Principles of distributed database systems*. Prentice-Hall, Inc., Upper Saddle River, NJ, USA, 1991.

[19] C. Reynaud, J.-P. Sirot, and D. Vodislav. Semantic integration of xml heterogeneous data sources. In M. E. Adiba, C. Collet, and B. C. Desai, editors, *IDEAS*, pages 199–208. IEEE Computer Society, 2001.

[20] K.-U. Sattler, I. Geist, and E. Schallehn. Concept-based querying in mediator systems. *The VLDB Journal*, 14(1):97–111, 2005.

[21] K.-D. Schewe. Fragmentation of object oriented and semistructured data. In *BalticDB&IS*, pages 253–266, 2002.

[22] F. V. Silveira and C. A. Heuser. Fragmentation and query decomposition in xml. Technical report, UFRGS, Brazil, April 2006. Available from http://www.inf.ufrgs.br/∼heuser/papers/repfv.pdf.

[23] T. W. W. W. C. (W3C). $http://www.w3.org/$.

[24] G. Wiederhold. Mediators in the architecture of future information systems. *Computer*, 25(3):38–49, 1992.