The main objective of a mediator-based data integration system is to provide a unified view of several distributed and heterogeneous data sources. This view, called mediation schema, corresponds to a set of elements computed from data available on the local data sources. One of the biggest challenges facing a data integration system consists in answering a user query submitted in terms of the mediation schema given that the data is at the sources. This problem consists mainly in reformulating the query in terms of the source schemas and in integrating the corresponding answers. In this paper we address the problem of query reformulation in the context of an XML-based data integration system. Our solution is based on algorithms that generate a set of source queries considering a given user query and a set of mappings between the mediation schema and the data source schemas.

ABSTRACT

The main objective of a mediator-based data integration system is to provide a unified view of several distributed and heterogeneous data sources. This view, called mediation schema, corresponds to a set of elements computed from data available on the local data sources. One of the biggest challenges facing a data integration system consists in answering a user query submitted in terms of the mediation schema given that the data is at the sources. This problem consists mainly in reformulating the query in terms of the source schemas and in integrating the corresponding answers. In this paper we address the problem of query reformulation in the context of an XML-based data integration system. Our solution is based on algorithms that generate a set of source queries considering a given user query and a set of mappings between the mediation schema and the data source schemas.

Categories and Subject Descriptors H.2 [Database Management]: Systems – query processing.

General Terms Languages.

Keywords Data integration, XML, query reformulation, mediation.

1. INTRODUCTION

Integrated access to distributed data is an important problem faced in many scientific and commercial domains. One of the main challenging problem in the context of data integration is query processing. In order to answer a query submitted to a data integration system various steps are necessary, including query decomposition, query translation, query optimization and data integration. The complexity of the query execution process depends mainly on the approach adopted to define the mappings between the mediation schema and the source schemas. Two main approaches have been used to specify such mappings: Global as View (GAV) and Local as View (LAV) [12, 6]. In the GAV approach each object of the global schema is expressed as a view (i.e. a query) on the data sources. On the other hand, in the LAV approach, mediation mappings are defined in an opposite way; each object in a given source is defined as a view on the global schema.

In this paper we address the problem of how to process a query submitted to a data integration system based on the GAV approach. The query execution process was developed as part of an XML-based data integration system, called Integra, proposed in [8]. We focus our discussion on the problem of how to decompose a user query into a set of remote data source queries using a set of mediation mappings. The remainder of this paper is organized as follows. Section 2 provides some definitions. Section 3 describes the query execution process. Section 4 explains our approach to query reformulation. Section 5 presents some implementation issues. Section 6 discusses some related works and Section 7 presents our conclusions.

2. SOME DEFINITIONS

To facilitate the query execution process and other relevant tasks of the Integra system a conceptual data model, called X-Entity [9], and a declarative internal language called XEQ (X-Entity Query language) [3] were proposed. In this section we give an overview of the main concepts proposed in the X-Entity model and in the XEQ language. We also present the formalism adopted to represent the mappings between the mediation schema and the source schemas.

2.1 X-Entity Model

Integra uses XML as the common data model for data exchange and integration. To represent the mediation schema and the source schemas Integra adopts the XML Schema language. To provide a high-level abstraction for information described in an XML schema a conceptual data model, called X-Entity model [9], was proposed. The X-Entity model is not a new formalism for conceptual modeling, rather it is an extension of the Entity Relationship model [2], i.e., it uses some basic features of the ER model and extends it with some additional ones to better represent XML schemas. The main concept of the X-Entity model is the entity type, which represents the structure of XML elements composed by other elements and attributes. In the X-Entity model, relationship types represent element-subelement relationships and
references between elements. In the following, we present the main X-Entity model constructs used to create X-Entity schemas. An X-Entity schema $S$ is denoted by $S = (E, R)$, where $E$ is a set of entity types and $R$ is a set of relationship types.

- **Entity type**: an entity type $E$, denoted by $E(A_1,\ldots,A_n,\{R_1,\ldots,R_m\})$, is made up of an entity name $E$, a set of attributes $A_1,\ldots,A_n$ and a set of relationships $R_1,\ldots,R_m$. An entity type represents a set of elements with a complex structure, composed of attributes and other elements (called subelements). An instance of an entity type is a particular element in the source XML document. Each entity type has attributes $A_1,\ldots,A_n$ that describe it. An attribute $A_i$ represents either an attribute or a subelement, which is not composed by other elements or attributes. In X-Entity diagrams, entity types are displayed as rectangles.

- **Containment relationship**: a containment relationship between two entity types $E$ and $E_i$ specifies that the entity type $E_i$, for example, has two attributes: $\langle$book$\rangle$ and $\langle$publisher$\rangle$, and each one of the entities being queried is identified and referred to an X-Entity schema. The SELECT element has one attribute, called ENTITY, whose value is the name of the entity type $E$ being queried. The SELECT element may be composed by other elements, as: i) a REFERENCE element that defines the join condition used to combine related elements from the entity type $E$ and a referenced entity type $E'$, ii) a $\langle$ATTRIBUTE$\rangle$ element that have in their content the name of an attribute of the entity type $E$ whose value is to be retrieved by the query, iii) a WHERE element that filters the data to be retrieved by the query. To illustrate the use of the XEQ internal language we present, in Figure 1, an XQuery query and its corresponding XEQ expression.

**User query**: Retrieve the title, edition, the title of the chapters and the publisher name of all books.

**XQuery**

```xml
<books>
  for $b$ in //book
    return <book>...
</books>
```

**XEQ**

```xml
<EXP>
  <SELECT ENTITY="books"/>
  <ATTRIBUTE title="ATTRIBUTE"/>
  <ATTRIBUTE year="ATTRIBUTE"/>
  <SELECT ENTITY="publisher$2"/>
  <ATTRIBUTE publisher_name="ATTRIBUTE"/>
  <REFERENCE NAME="ref_book_publisher"/>
  <MAP ATTRIBUTE="ref_book_publisher"/>
</EXP>
```

**Figure 1. User query example**

In the example of Figure 1, consider the X-Entity schema $S_2$ of Figure 2. During the query translation to XEQ the query is parsed and each one of the entities being queried is identified and represented using a SELECT element. In the example above, $\langle$book$\rangle$ is the entity whose values should be retrieved. Since $\langle$chapter$\rangle$ is a subentity of $\langle$book$\rangle$, to retrieve such information it should be used a nested SELECT element. It is important to observe that in the SELECT element associated with the referenced entity $\langle$publisher$\rangle$, there is a MAP element. This element indicates how the join operation between the entity $\langle$book$\rangle$ and the

- **Reference relationship**: a reference relationship, denoted by $R(E_i, E_2,\{A_{i1},\ldots,A_{in}\},\{A_{21},\ldots,A_{2n}\})$, where $R$ is the relationship name and $\langle$min, max$\rangle$ defines the minimum and the maximum number of instances of $E_i$ that can be associated with an instance of $E$. In X-Entity diagrams, containment relationships are displayed as diamond-shaped boxes labeled with contains. The straight lines connecting the relationship with the participating entities are directed from the entity $E$ to the entity $E_i$.

- **Containment relationship**: a containment relationship between two entity types $E$ and $E_i$ specifies that each instance of $E$ contains instances of $E_i$. It is denoted by $R(E, E_i, \langle$min, max$\rangle)$, where $R$ is the relationship name and $\langle$min, max$\rangle$ defines the minimum and the maximum number of instances of $E_i$ that can be associated with an instance of $E$. In the following, we present the main X-Entity model constructs used to create X-Entity schemas. An X-Entity schema $S$ is denoted by $S = (E, R)$, where $E$ is a set of entity types and $R$ is a set of relationship types.

- **Entity type**: an entity type $E$, denoted by $E(A_1,\ldots,A_n,\{R_1,\ldots,R_m\})$, is made up of an entity name $E$, a set of attributes $A_1,\ldots,A_n$ and a set of relationships $R_1,\ldots,R_m$. An entity type represents a set of elements with a complex structure, composed of attributes and other elements (called subelements). An instance of an entity type is a particular element in the source XML document. Each entity type has attributes $A_1,\ldots,A_n$ that describe it. An attribute $A_i$ represents either an attribute or a subelement, which is not composed by other elements or attributes. In X-Entity diagrams, entity types are displayed as rectangles.

- **Containment relationship**: a containment relationship between two entity types $E$ and $E_i$ specifies that each instance of $E$ contains instances of $E_i$. It is denoted by $R(E, E_i, \langle$min, max$\rangle)$, where $R$ is the relationship name and $\langle$min, max$\rangle$ defines the minimum and the maximum number of instances of $E_i$ that can be associated with an instance of $E$. In X-Entity diagrams, containment relationships are displayed as diamond-shaped boxes labeled with contains. The straight lines connecting the relationship with the participating entities are directed from the entity $E$ to the entity $E_i$.

- **Reference relationship**: a reference relationship, denoted by $R(E_i, E_2,\{A_{i1},\ldots,A_{in}\},\{A_{21},\ldots,A_{2n}\})$, where $R$ is the relationship name and $\langle$min, max$\rangle$ defines the minimum and the maximum number of instances of $E_i$ that can be associated with an instance of $E$. In X-Entity diagrams, containment relationships are displayed as diamond-shaped boxes labeled with contains. The straight lines connecting the relationship with the participating entities are directed from the entity $E$ to the entity $E_i$.

- **Containment relationship**: a containment relationship between two entity types $E$ and $E_i$ specifies that each instance of $E$ contains instances of $E_i$. It is denoted by $R(E, E_i, \langle$min, max$\rangle)$, where $R$ is the relationship name and $\langle$min, max$\rangle$ defines the minimum and the maximum number of instances of $E_i$ that can be associated with an instance of $E$. In X-Entity diagrams, containment relationships are displayed as diamond-shaped boxes labeled with contains. The straight lines connecting the relationship with the participating entities are directed from the entity $E$ to the entity $E_i$.

- **Reference relationship**: a reference relationship, denoted by $R(E_i, E_2,\{A_{i1},\ldots,A_{in}\},\{A_{21},\ldots,A_{2n}\})$, where $R$ is the relationship name and $\langle$min, max$\rangle$ defines the minimum and the maximum number of instances of $E_i$ that can be associated with an instance of $E$. In X-Entity diagrams, containment relationships are displayed as diamond-shaped boxes labeled with contains. The straight lines connecting the relationship with the participating entities are directed from the entity $E$ to the entity $E_i$.

- **Reference relationship**: a reference relationship, denoted by $R(E_i, E_2,\{A_{i1},\ldots,A_{in}\},\{A_{21},\ldots,A_{2n}\})$, where $R$ is the relationship name and $\langle$min, max$\rangle$ defines the minimum and the maximum number of instances of $E_i$ that can be associated with an instance of $E$. In X-Entity diagrams, containment relationships are displayed as diamond-shaped boxes labeled with contains. The straight lines connecting the relationship with the participating entities are directed from the entity $E$ to the entity $E_i$.
entity publisher should be done in order to retrieve the value of the publisher_name attribute.

2.3 Correspondence assertions

Our approach to query reformulation is widely based on correspondence assertions [11] specifying relationships between the mediation schema and the source schemas. In this section, we describe just the correspondence assertions relevant to this paper. Other correspondence assertions may be found in [8].

Definition 1 (Entity correspondence assertion): Let $E_1$ and $E_2$ be entity types. The correspondence assertion $E_1 \equiv E_2$ specifies that $E_1$ and $E_2$ are semantically equivalent, i.e. they describe the same real world concept and they have the same semantics.

Definition 2 (Attribute correspondence assertion): Let $A_1$ be an attribute of the entity type $E_1$ and $A_2$ an attribute of the entity type $E_2$, such that $E_1 \equiv E_2$. The correspondence assertion $A_1A_2$ specifies that $A_1$ and $A_2$ are semantically equivalent (correspond to the same concept in the real world).

Definition 3 (Link): Let $X_i$ and $X_j$ be elements of an X-Entity schema (an element can be an entity type, a containment relationship type or an attribute). $X_iX_j$ is a link if: i) $X_i$ is an attribute of the entity type $X_j$, or ii) $X_j$ is an entity type participating in the relationship type $X_i$ (or vice-versa).

Definition 4 (Path): If $X_1, \ldots, X_n$ are elements of a schema, such that $\forall X_i, 1 \leq i \leq n, X_iX_{i+1}$ is a link, then $X_1X_2 \ldots X_n$ is a path from $X_1$ (the path is directed from $X_1$ to $X_n$).

Definition 5 (Inverse path): If there is a path from $X_i$ denoted by $X_1X_2 \ldots X_n$, such that $X_n$ is an entity type, then there is a path denoted by $(X_nX_{n-1} \ldots X_1)^{-1}$, which is called the inverse path of $X_1X_2 \ldots X_n$.

Definition 6 (Path correspondence assertion): Let $P_1$ and $P_2$ be two paths: Case 1: $P_1 = X_1X_2 \ldots X_n$, and $P_2 = Y_1Y_2 \ldots Y_m$, where $X_i \equiv Y_j$. The correspondence assertion $P_1 \equiv P_2$ specifies that the entity types $X_n$ and $Y_m$ are semantically equivalent. Case 2: $P_1 = X_1X_2 \ldots X_n$, and $P_2 = (Y_1Y_2 \ldots Y_m)^{-1}$, where $X_i \equiv Y_j$. The correspondence assertion $P_1 \equiv P_2$ specifies that the entity types $X_n$ and $Y_j$ are semantically equivalent.

To better illustrate our ideas, consider the mediation schema and the source schemas presented in Figure 2.

Figure 2. Mediation schema $S_{med}$ and source schemas $S_1$ and $S_2$.

Table 1 presents the correspondence assertions (CA) between $S_{med}$ and the source schemas $S_1$ and $S_2$.

![Figure 2](image)

Table 1. Correspondence assertions between the mediation schema $S_{med}$ and the source schemas $S_1$ and $S_2$

<table>
<thead>
<tr>
<th>CA</th>
<th>$S_{med}$</th>
<th>$S_1$</th>
<th>$S_2$</th>
</tr>
</thead>
<tbody>
<tr>
<td>CA1</td>
<td>book.m == book1</td>
<td>book1</td>
<td>book.m</td>
</tr>
<tr>
<td>CA2</td>
<td>book.m.title.m == book.title</td>
<td>book.title</td>
<td>book.m.title.m</td>
</tr>
<tr>
<td>CA3</td>
<td>book.m.publisher.m == book.publisher</td>
<td>book.publisher</td>
<td>book.m.publisher.m</td>
</tr>
<tr>
<td>CA4</td>
<td>chapter_m == chapter1</td>
<td>chapter1</td>
<td>chapter_m</td>
</tr>
<tr>
<td>CA5</td>
<td>chapter_m.chapter_title_m == chapter.chapter_title</td>
<td>chapter.chapter_title</td>
<td>chapter_m.chapter_title_m</td>
</tr>
<tr>
<td>CA6</td>
<td>book.m.chapter_m.chapter_ref_book.m == (chapter1.chapter_ref_book.book1)</td>
<td>chapter1.chapter_ref_book.book1</td>
<td>book.m.chapter_m.chapter_ref_book.m</td>
</tr>
<tr>
<td>CA7</td>
<td>book.m == book2</td>
<td>book2</td>
<td>book.m</td>
</tr>
<tr>
<td>CA8</td>
<td>book2.m.title.m == book2.title</td>
<td>book2.title</td>
<td>book2.m.title.m</td>
</tr>
<tr>
<td>CA9</td>
<td>chapter_m == chapter2</td>
<td>chapter2</td>
<td>chapter_m</td>
</tr>
<tr>
<td>CA11</td>
<td>book2.m.chapter_m.chapter_title_m == chapter2.chapter.chapter_title</td>
<td>chapter2.chapter.chapter_title</td>
<td>book2.m.chapter_m.chapter_title_m</td>
</tr>
<tr>
<td>CA12</td>
<td>book2.m.publisher.m == book2.publisher</td>
<td>book2.publisher</td>
<td>book2.m.publisher.m</td>
</tr>
</tbody>
</table>

3. QUERY EXECUTION PROCESS

The query execution consists in receiving a user query, expressed in a declarative query language, and in returning a query answer expressed in XML as output. The query execution process [3], described in Figure 3, may be summarized as follows: i) The mediator receives a user query $Q$ and performs the necessary translation and reformulation to produce the set of local XEQ expressions ($q_1$, $q_2$, ..., $q_n$). ii) Next, the mediator sends these expressions to the corresponding wrappers which translate them to the source query language producing the local queries ($q_1$, $q_2$, ..., $q_n$). iii) The wrappers receive answers for such queries ($r_1$, $r_2$, ..., $r_n$) from the data sources, these answers are translated to XML and sent to the mediator and iv) Finally, the mediator integrates the XML answers ($r_1$, $r_2$, ..., $r_n$) and returns the integrated result $R$ as the answer to the original user query $Q$. The components responsible for the query execution process are presented in Figure 3 and described next.

![Figure 3](image)
In this section, we describe the basic algorithm that reformulates a query expression over the mediation schema. The ReformulateQuery algorithm takes as input the XML data received from the wrappers and converts them to the mediation schema. This translation is necessary to facilitate the further data integration process. Transformation functions are used to convert data format to be conform to the mediation schema. The Data Integrator takes as input a set of XML documents and transforms them into a single integrated document.

4. QUERY REFORMULATION

In this section, we describe the basic algorithm that reformulates a XEQ expression over the mediation schema into a set of XEQ expressions over the source schemas. Let \( Q \) be the original user query and \( X_Q \) be the expression obtained after the translation of \( Q \) to the XEQ representation. There are three phases in the algorithm:

i) Identification of the mediation entity \( E_m \) being queried: this entity is specified in the most external SELECT element of the XEQ expression.

ii) Identification of data sources relevant to answer the user query: a data source \( DS_i \) is relevant to answer the user query \( Q \) if it contains one or more source entities necessary to compute the mediation entity \( E_m \), i.e., source entities semantically equivalent to \( E_m \).

iii) Reformulation phase: this phase consists in generating a XEQ expression for each one of the data sources relevant to answer \( Q \). This is done using the algorithms presented in Figure 4. Next consider:

- \( T/@ENTITY \): value of the ENTITY attribute of the SELECT element \( T \)
- \( T/@ATTRIBUTE \): value of an ATTRIBUTE element of the SELECT element \( T \)
- \( T/\text{WHERE} \): value of a WHERE element of the SELECT element \( T \)
- \( A/text( \) \): retrieves the content of the ATTRIBUTE element \( A \)
- \( ECA(S_i, E_m) \): retrieves the entity correspondence assertion \( C \equiv E_m \), where \( E_j \in S_i, E (S_i, E \) is the set of entity types of the source schema \( S_i \)
- \( ACA(S_i, A_m) \): retrieves the attribute correspondence assertion \( C \equiv A \), where \( A_j \in E' \) and \( E' \subseteq S_i, E (E.A \) is the set of attributes of the entity type \( E' \)
- \( X\#F \): concatenation operator that joins two XEQ expressions \( X_1 \) and \( X_2 \)

During the third phase of the query reformulation process the ReformulateQuery algorithm is executed for each one of the relevant data sources \( DS_1, \ldots, DS_n \). The first task of this algorithm consists in reformulating the element \( X/\text{Exp}/\text{SELECT} \) that defines the mediation entity \( E_m \) being queried. To do this, the algorithm calls the procedure ReformulateSelect. This procedure recursively analyses the nested SELECT elements of \( T (T = X/\text{Exp}/\text{SELECT}) \) in order to identify the subentities \( E_j \) and attributes \( A_j \) of the mediation entity \( E_m \) such that \( T/@ENTITY = E_m \). For each mediation entity \( E_j \) the algorithm ReformulateSelect obtains the correspondence assertion which specifies how to compute \( E_j \) over the data source \( DS_i \). In the same way for each attribute \( A_j \) the algorithm obtains the correspondence assertion which specifies how to compute \( A_j \) over the data source \( DS_i \). The algorithm also analyses WHERE elements to transform the constraints applied on the mediation entities into constraints on the source entities. This transformation is done using the algorithm ReformulateWhere. XEQ elements of the resulting source expression are created by the algorithm AddPathToQuery. Due to space limitations, in this paper we present just the ReformulateQuery and ReformulateSelect algorithms. More information about the other algorithms may be found in [3].

Algorithm ReformulateQuery(XQ, S) (S is the X-Entity schema of the data source \( DS_i \))

Lines 1-7 were generated based on the correspondence assertions \( CA_1 \) and \( CA_2 \), presented in Figure 5. As specified by the most external SELECT element of \( I \) we have that the entity being queried is \( book_m \) (X/Exp/Select/@ENTITY = book_m). Next, based on the set of correspondence assertions presented in Table 1 the algorithm identifies the relevant data sources to compute \( book_m \). According to the correspondence assertions \( CA_1 \) and \( CA_2 \), \( book_m \) is the X-Entity schema of the data source \( DS_i \).

Figure 4. Query reformulation Algorithms

User query \( Q \): Retrieve the title, publisher and the title of the chapters all books.

Figure 5. User query

In the first step, the algorithm generates the source expression \( X_S1 \) presented in Figure 5, for the data source \( DS_1 \). The SELECT element of line 2 was generated from the correspondence assertion \( CA_1 \). Lines 3 and 4 were generated based on the correspondence assertions \( CA_2 \) and \( CA_3 \). To compute the subentity \( chapter_m \) the algorithm created a nested select element (line 5) according to the correspondence assertion \( CA_4 \). As chapter, is
focused on the problem of how to process a user query submitted to an XML-based data integration system. To translate queries from XEQ to the data sources’ native language, we used two real databases that stores medical information: Healthnet and TELEMED, both implemented in MySQL. In the current version of the prototype, the mediation schema and the correspondences between the mediation schema and the source schemas are manually defined and stored in the Mediator Knowledge Base. In the Integra prototype, the query is translated to SQL using a XSLT stylesheet, and it is submitted to the data source through a JDBC driver.

6. RELATED WORK

The work presented in [4] states and solves the query decomposition problem for XML publishing in a general setting. It allows mixed (XML and relational) storage of proprietary data and exploits redundancies (materialized views, indexes and caches) to enhance performance. Other problem faced during query processing consists in translating data source queries to the native query language. To improve the query translation process, data integration systems use languages based on scripts or templates as the input format for wrappers [7]. There are also systems that use XML query languages as input format for wrappers. The e-XML [5], MARS [4] and XPERANTO [10] systems provide integration of multiple data sources based on XML schemas. These data sources are accessed through XQuery. AutoMed [1] adopts a functional language, called IQL, to provide a common query language where queries written in high level query languages can be translated into and out of. In our approach, the user query may be specified in a high level query language. This query is translated to XEQ and it can be easily rewritten into a set of local subqueries. As XEQ is an XML-based language we can take advantage of XML benefits as the capability to easily transform XML data into different formats and to easily navigate through the hierarchical structure of XML data.

7. CONCLUSION

In this paper, we focused on the problem of how to process a user query submitted to an XML-based data integration system. To solve this problem we proposed X-Entity, an ER-based conceptual data model, and XEQ, an XML-based language, whose specification facilitates both query decomposition and translation. The process of decomposing or translating a XEQ expression to other query languages consists in performing a recursive navigation that visits the expression elements and constructs the query result. In this paper we did not consider the problem of generating a query execution plan nor the problem of query optimization. As a future work, we intend to study such problems in order to improve our proposed approach.

8. ADDITIONAL AUTHORS

Juliano F. de Souza, jsf@cin.ufpe.br, Center for Informatics Federal University of Pernambuco, P.O. Box 7851 Cidade Universitária, Recife - PE, Brazil, 50732-970, +55 81 2126.8430

9. REFERENCES


1 http://www.nutes.ufpe.br/servicos/health.html
2 Database of a brazilian hospital (Hospital Português – Recife)