Agora: Living with XML and Relational

Ioana Manolescu, Daniela Florescu, Donald Kossmann, Florian Xhumari, Dan Olteanu*
{Ioana.Manolescu, Daniela.Florescu, Florian.Xhumari, Dan.Olteanu}@inria.fr
kossmann@informatik.tu-muenchen.de

1 Introduction

There has been a significant body of research in the last fifteen years dedicated to integration of data from various repositories, exhibiting heterogeneous formats, and sometimes access restrictions; for a survey of such systems see, for example, [12]. The main technical issues to be addressed in a mediation system are: how to semantically unify heterogeneous data formats and schemas, and how to use query processing capabilities of participant data sites and that of the mediator in order to answer a particular query.

Systems like the Information Manifold, and Garlic from IBM have chosen the relational and respectively the object-oriented model as the integration model. Given the popularity of XML as a data description format, more and more DBMS manufacturers have added to their systems the capability to export relational or object-oriented data to an XML format; other data formats (flat data files, regular HTML, PowerPoint presentations, annotated text) are also easily converted to XML. XML has become somehow a *de facto* standard for information exchange.

While XML has clear advantages as a description format, state-of-the-art query optimization and query processing algorithms for data integration still rely on the relational model. This is the case, for example, of the existing algorithms for answering queries using views [9]. Given the richness of the semistructured model (and the peculiarities of XML as a data model), algorithms of equivalent efficiency and ease-of-use, but *designed for* XML, are more difficult to find. Recent projects like [1] and [4] put XML in the center of query processing, describing data sources in XML and evaluating queries over a tree-structured model; optimization is ignored or reduced to a few simple heuristics. As a consequence, performance might degrade whenever an XML data source joins a set of relational sources, that we were able to integrate efficiently.

The Agora System. The particularity of our data integration system is that it employs XML as the user interface format, while all data flows inside the query processor consist of relational tuples. Queries are posed using an XML query language and the results are formatted as XML documents, making the underlying relational engine transparent to the user.

Agora is implementated on top of the Le Select data integration system [8], developed in the Caravel project, at INRIA Rocquencourt. Our goal in designing Agora was to investigate the feasibility and the attainable performance of a system that processes XML queries based on relational technology. Besides being intelectually interesting, we find this approach particularly tempting, given the strength acquired in the research and industrial communities in the field of query processing for relational data. We demonstrate the particular techniques that we have implemented in Agora to complement Le Select's functionalities, namely:

- how to define a generic, relational, virtual integration schema, that describes the content of XML documents
- how to translate queries from a query language dedicated to XML to the relational integration schema, and how to rewrite the resulting query using view definitions that describe the XML documents
- how query optimization extends to cope with access pattern limitations, when constructing query plans over heterogeneous sources
- how a text index (implemented by a few relational tables) improves performance and helps formulating user queries over the semistructured part of the data.

2 Adding XML Value into a Relational Data Integration System

We will now briefly describe the architecture of the Le Select relational integration system, that provides the relational framework that Agora is based on; we will then

*The permanent address of this author is the CS Department of the Polytechnic University of Bucharest, Romania. This work was done while the author was in INRIA Rocquencourt

Permission to copy without fee all or part of this material is granted provided that the copies are not made or distributed for direct commercial advantage, the VLDB copyright notice and the title of the publication and its date appear, and notice is given that copying is by permission of the Very Large Data Base Endowment. To copy otherwise, or to republish, requires a fee and/or special permission from the Endowment.

Proceedings of the 26th VLDB Conference, Cairo, Egypt, 2000.

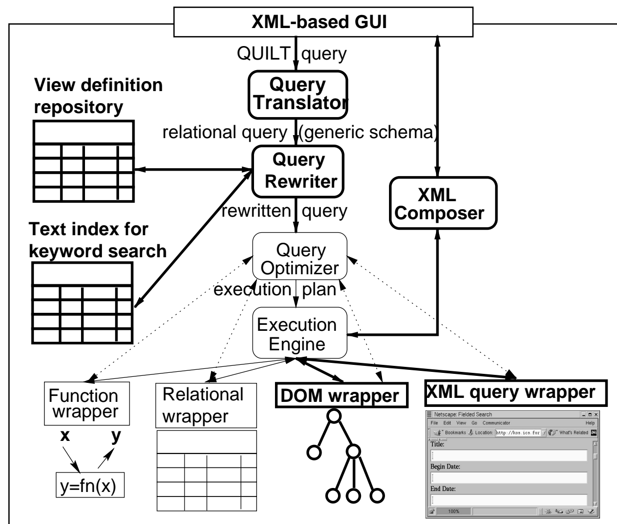

Figure 1: General architecture of the Agora system.

detail our technical contributions. Agora, as well as Le Select, is meant to function on several servers running identical code, each server owning and sharing data and programs. The complete architecture of a single Agora server is depicted in Figure 1. The components inherited from Le Select are detailed in 2.1 and are shown in the figure in thin lines; the novel components that we add are described under 2.4 and are shown with bold lines and fonts. Solid lines describe data flow during query processing; dotted lines represent the flow of statistic informations provided to the optimizer by the data and program wrappers.

2.1 Le Select

This system offers a framework for publishing and querying relational data and programs. A network of connected servers share their data and programs and collaborate in answering users queries posed against any server. Relational data and programs are published via specific wrappers. For some frequent data formats, like native relational and formatted files data, pre-defined wrappers can be easily configured. Query optimization is done on the site the query, while query execution is distributed among the sites. Wrappers export statistics about their data, like cardinalities, available access patterns, and, for functions, cost of executing them. They also export information about their query processing capabilities, in terms of evaluation of arithmetic expressions, capacity of performing a join, an equality test etc.

The query optimizer of the query site consults the wrapper capabilities and constructs an execution plan that distributes the work to be done among the wrappers of the data and program sources, their corresponding sites, and the site of the query. The optimizer is cost-based; repartitioning of tasks on different locations is made with the goal of minimizing the global cost (an important part of which is the cost of data transfers). The optimizer has a built-in treatment of access patterns, in order to deal with access restrictions and with functions, as described in [7]; this makes Le Select a suitable candidate for extension to XML. The execution engine contains an efficient implementation of bind joins, that minimizes data transfers, if the size of the attribute(s) to be passed across the join is important. More details on its functioning can be found under [8].

2.2 Motivating Example.

We consider integrating nutritional information (what food ingredients are recommended/forbidden for people with a certain ailment) with patient medical records and a collection of cooking recipes, to help a nutritional expert advise his patients on how and what to eat. Independent of the native storage, all information is available via an XML interface.

The nutritional information is stored in a relational database; the simplified relational schema consists of two tables:

Recommendations(illness, recFood)
Interdictions(illness, intFood)

The relational data is available to the data integration system as a virtual XML source. We use for this purpose a mapping like the one described in the Xperanto project [3]. Hence, this data will appear to the user as the following (virtual) XML file:

```
<Recommendations>
    <tuple>
        <illness>some illness</illness>
        <recFood>some food</recFood>
    </tuple>
    ...
</Recommendations>
<Interdictions>
    <tuple>
        <illness>some illness</illness>
        <intFood>some food</intFood>
    </tuple>
    ...
<Interdictions>
```

The structure of the patients medical records tends to be irregular, and these records are stored in a proprietary format. The logical interface that they export to the data integration system is also based on XML, as in the example shown below, and the only access methods it supports are based on the Document Object Model (DOM) API:

```
<patient_rec patientID="1000" sex="M">
    <pers_rec>
        <dob>12/6/1965</dob>
        <illness>Calcium deficiency</illness>
        <ilness>Pneumonia in 1969</illness>
    </pers_rec>
</patient_rec>
```

Finally, the cooking recipes can be obtained from a website that only allows searching its database by keyword search over the ingredient field of the recipes. As an example, a query searching for recipes using salmon as an ingredient produces as a result the following XML document:

```
<recipe recID="epicurious124">
    <plate>"Potato and salmon casserole"</plate>
    <ingredient qty="14oz">pink salmon, drained</>
    <ingredient qty="2 1/2 pounds">russet potatoes</>
    <ingredient qty="1/2cup">chopped green onions</ >
```

```
    <ingredient qty="2 pc">large eggs</>
    <directions>Preheat oven to 400F. Separate salmon>
        into chunks...</>
</recipe>
```

A data integration query can extract and combine data from all three sources. As stated before, the query interface visible to the users, and, subsequently, the query language are completely based on XML. In the absence of a standard XML query language, we will use in the demonstration (a subset of) the Quilt query language [10]. To get a first glimpse at the language, consider the following query, asking for the medical records of all patients whose personal records contain "illness" and "calcium defficiency":

```
for $p in document("medicalRecords.xml")//patient_rec
where contains($p,"calcium deficiency") and
    contains($p,"illness")
return $p
```

In a Quilt query, the for clause binds variable by iterating over collections of XML nodes, the where clause specifies selection conditions (much like in traditional query languages like OQL and SQL), and the return clause constructs the result (this can involve construction of element hierarchies etc).

2.3 Modeling XML data sources

Generic Relational Schema. The particularity of our system resides in the fact that even if the *external* data model (i.e. as seen by the user formulating the queries) is XML, the *internal* data model (i.e. used for the data flowing inside the execution engine) is still relational. Hence, the first step to acomplish is to map the XML view of the data into the relational model. XML data sources are modeled in our system by a generic relational schema [5], which is independent of any particular XML data instance. The tables that we propose to add are shown in figure 2.3. The tables Element, ElemContent, Attribute, ElemAttribute, Tag and Value fully describe the contents of the XML elements. The Value table stores all the string values found in the document, either attribute values or contents of text nodes.

Describing Access and Storage by Views. The generic relational schema is used to logically describe the content of data sources; in order to specify the actual storage and the access patterns supported by the sources, Agora allows for defining views with binding patterns (in a manner similar to that described in [11]. The view definitions can be arbitrarily complex, and by consulting them, the query processor is informed of the alternative ways to access the data, as well as the costs involved. As an example, let us consider the DOM method call retrieving all elements with a specific tag within a given document:

{y} = x.getElementByName(z)

This API call can be modeled as a view over the generic relational schema. Note that since the document and the tag must be known in order to make this call, there are inherent constraints on consulting this view, that we model by the view's binding pattern (note that the document and the tag are bound variables in the view definition):

V(x^b,y^f,z^b):-ElemDoc(y,x^b),Element(y,t),Tag(t,z^b)

The advantage of the logical modelisation of XML by a collection of generic relational tables is twofold. First, these tables can be added to any relational mediation system, regardless of its mechanism for defining a global schema (or even if there is no such schema). Second, arbitrarily complex structures and physical access methods can be easily described as views with binding patterns over this generic schema, as in the example we have shown; thus, the system can take full advantage of optimized access paths, materialized views etc.

2.4 Query Processing Methodology.

We sketch here the main steps of a query scenario with our system.

Formulating the Quilt query. There are two ways of posing a query to the system. Expert users that are familiar with the data structure can write full Quilt queries, while novice users can use the GUI that allows for browsing the data, and progressively refine their query, exclusively via the GUI. In the case of our sample application, a novice user would have to discover, for example, the structure of recipe and Recomandation elements and pose the join condition between the two.

Translating the Quilt query into SQL. The Quilt query will be then translated into a set of correlated, parameterized SQL queries over the relational generic schema. These queries are equivalent with the original Quilt query; instead of producing as result an XML document they produce the equivalent instance of the generic relational schema. For example, the sample Quilt query shown in section 2.2 can be rewritten into the following equivalent one[1]:

```
calciumDefPatient(patRec):-Document(docID,_)
    ElemDoc(docID, patRec),ElemDoc(docID,e2),
    Element(e1,t1),Tag(t1,"patient_rec"),
    ElemContent(e1,e2,null,_,Element(e2,t2),
    Tag(t2,"pers_rec"),contains(e2,"calcium",depth,tag),
    contains(e2,"deficiency",depth,tag),
    contains(e1,"illness",depth2,tag2).
```

The support for this translation phase is an underlying algebraic model close to the one described in [2].

Query rewriting using views. The relational query obtained in the previous step is then rewritten into an equivalent relational query which uses only the views modeling the real access patterns to the native XML data, as well as to the actual access path to relational data (that was referred to under its XML interface in the query). We use a simple rewriting query using views algorithm which produces equivalent rewritings of the relational query, with respect to bag semantics[9].

Rewriting the previous query using the view V thus defined would yield:

```
Result(patRec):-Document(docID,_),
    V(docID,patRec,"patient_rec"),
    ElemContent(e1,e2,null,_),V(docID,e2,"pers_rec"),
    contains(e2,"calcium",depth,tag),
    contains(e2,"deficiency",depth,tag).
```

[1] We use Datalog instead of SQL for simplicity.

Document	(**docID**, docURL)	Element	(**elID**, tagID)
Value	(**valID**, value)	ElemContent	(**parentID, childID**, valID, index)
Tag	(**tagID**, valID)	ElemAttribute	(**elID**, attID, valID)
Attribute	(**attID**, valID, type, isRequired)	ElemDoc	(**elemID, docID**)
Word	(**wordID**, word)	Contains	(elID, wordID, depth, tag)

Figure 2: Generic relational schema

Support for Keyword Search. We now explain the purpose of the last two generic tables, Word and Contains. The user might not know the particular structure of the documents or might have only some partial knowledge about it. For example, when querying the medical records for calcium-deficient patients, she might ignore if names of the maladies are to be found under illness tag, nested within the consultation tag, or directly under the pers_rec tag.

A text index at the granularity of XML elements can be used to retrieve all the XML elements that contain the words "calcium deficiency", at a specific nesting depth, in the content (as opposed to in the data tags). This index can be used as a help for novice users to "browse" the information content available, or as a filter for more structured queries. For the purpose of query optimization and execution, this type of index will be modeled as a relational table (see 2.3) with binding patterns limitations. More details on the usage and possible implementations of such an XML index can be found in [6].

Query Optimization and Execution. The rewritten query is optimized in a cost-based manner, following the optimization principles of LeSelect, that we described in 2.1; see also [7]. The result of the execution is a set of tuples.

Assembling the Result in XML. The tuples thus obtained are then grouped and organized into XML documents, that are presented to the user. We complete the illusion of an "all-XML" system.

3 Implementation and Scenarios

All the implementation (Le Select as well as the top layer for query rewriting, full-text indexing etc.) is done in Java; we use Oracle 8i as the DBMS that stores the text index and the index on metadata. The demonstration will be shown on a PC under Windows NT. Our scenario is close to the motivating example. We will use real life data collections, namely several collections of cooking recipes available on the Web (that we will replicate on the demonstration machine for the purpose of the demo), a relational source of nutritional information, and a set of XML medical files (corrupted for the sake of confidentiality). We plan to show:

- how to register the relational and XML data sources on several distict servers, and how to create instances of pre-defined wrappers for these sources;
- how to pose queries on one server, with the help of the GUI interface; how the full-text index is used to help the user discover the structure of available data and reformulate queries;
- how query rewriting operates using the available view definitions that describe XML and relational sources, by tracing the rewriting, optimization and execution of the same query;
- what is the influence of wrapper configuration and data localization on the final distributed execution of the query.
- what are the performances that can be achieved by the system.

The demonstration will build a case for what we consider to be an interesting, solid, and efficient alternative to pure-XML mediation.

References

[1] C. K. Baru, A. Gupta, B. Ludäscher, R. Marciano, Y. Papakonstantinou, P. Velikhov, and V. Chu. XML-based information mediation with MIX. In *Proc. of ACM SIGMOD Conf. on Management of Data*, pages 597–599, 1999.

[2] C. Beeri and Y. Tzaban. SAL: An algebra for semistructured data and XML. In *Proceedings of the International Workshop on the Web and Databases, Philadelphia, Pennsylvania*, 1999.

[3] M. Carey, D. Florescu, Z. Ives, Y. Lu, J. Shanmugasundaram, E. Shekita, and S. Subramanian. XPERANTO: Publishing object-relational data as XML. In *WebDB Workshop, in conj. with ACM Sigmod*, 2000.

[4] V. Cristophides, S. Cluet, and J. Simeon. On wrapping query languages and efficient XML integration. In *Proc. of ACM SIGMOD Conf. on Management of Data*, 2000.

[5] D. Florescu and D. Kossmann. Storing and querying XML data using an RDMBS. In *IEEE Data Engineering Bulletin*, volume 22(3), pages 27–34, 1999.

[6] D. Florescu, D. Kossmann, and I. Manolescu. Integrating keyword search into XML query processing. In *Proc. of the Int. WWW Conf.*, 2000.

[7] D. Florescu, A. Levy, I. Manolescu, and D. Suciu. Query optimization in the presence of limited access patterns. In *Proc. of ACM SIGMOD Conf. on Management of Data*, pages 311–322, 1999.

[8] http://www-caravel.inria.fr/Eaction_Le_Select.html.

[9] A. Y. Levy. Answering queries using views: a survey. submitted to publication, available at http://www.cs.washington.edu/homes/alon/.

[10] J. Robie, D. Chamberlin, and D. Florescu. The Quilt query language for semistructured data and XML. In *Proceedings of the International Workshop on the Web and Databases, Dallas, Texas*, 2000.

[11] O. G. Tsatalos, M. H. Solomon, and Y. E. Ioannidis. The GMAP: A versatile tool for physical data independence. In *Proc. of the Int. Conf. on Very Large Data Bases (VLDB)*, pages 367–378, Santiago, Chile, 1994.

[12] J. D. Ullman. Information integration using logical views. In *Proc. of the Int. Conf. on Database Theory (ICDT)*, Delphi, Greece, 1997.