XML-based Components for Federating Multiple Heterogeneous Data Sources

Georges Gardarin, Fei Sha, Tuyet-Tram Dang-Ngoc

XML-based Components for Federating Multiple Heterogeneous Data Sources

Georges Gardarin, Fei Sha, Tram Dang Ngoc

PRISM Laboratory, University of Versailles-St-Quentin
45 avenue des Etats-Unis, 78035 Versailles Cedex, FRANCE
E-mail: firstname.lastname@prism.uvsq.fr

Abstract. Several federated database systems have been built in the past using the relational or the object model as federating model. This paper gives an overview of the XMLMedia system, a federated database system mediator using XML as federating model, built in the Esprit Project MIRO-Web. The system is composed of four main components: a *wrapper generator* using rule-based scripting to produce XML data from various source formats, a *mediator* querying and integrating relational and XML sources, an *XML DBMS extender* supporting XML on top of relational DBMSs, and client tools including a *Java API* and an *XML query browser*. The results demonstrate the ability of XML with an associated query language (we use XML-QL) to federate various data sources on the Internet or on Intranets.

1 Introduction

In the past few years, research in semistructured data has become a hot topic [1], [2]. Different approaches have been proposed for efficiently managing semistructured data from diverse sources. Significant experiences are the Lore system [13] developed at Stanford University built from scratch for managing semistructured data, Ozone built on top of the O2 object-oriented DBMS, and STRUDEL [8] developed at AT&T for managing web-based data.

In this paper we present our experience in federating multiple data sources using XML and XML-QL, a query language jointly developed by AT&T and INRIA [4]. This work is part of the European ESPRIT project MIRO-Web [7], a federated database system integrating relational, object-oriented and semistructured data. MIRO-Web is federating heterogeneous data sources using a semistructured data model. The model is a slight variation of OEM [13]. Data are exchanged among sources using XML and XML-QL, i.e., a query language based on XML patterns [4]. As repository to store and retrieve XML documents, we use Oracle 8.0, the well known object-relational DBMS.

In the following, we give an overview of our federated database system architecture and describe the main components of the system. The system is now called XMLMedia (for Mediating in XML) and is on his way to be commercialized by a French-based company. Components include an *XML wrapper* using a rule-based data extraction technology developed at GMD IPSI in Germany, an *XML extender* (also called a cartridge) to store and query XML documents in a relational DBMS, an *XML mediator* which receives XML-QL queries from client sites, then decomposes them in either SQL or XML subqueries, routes the resulting subqueries to the source wrapper and assemble the results in XML. Finally, it also includes an *XML query browser* to formulate queries in XML-QL and present the results, plus a *Java DOM API* for submitting queries in Java and manipulating results.

The paper is organized as follows. Section 2 introduces the objectives of the XMLMedia system and gives some background on related projects. Section 3 describes the system architecture and introduces the components. Section 4 details the XML extender for relational DBMSs. Section 5 presents the principles of the mediator. We conclude in Section 6 and summarize the advantages of XML for mediation

2 Objectives and Backgrounds

In this section, we summarize the objectives of our system and present some backgrounds on semisructured databases.

2.1 System Objectives

The goal of the XMLMedia system is to provide in an efficient way integrated access to multiple data sources on the Internet or on intranets using XML protocols and tools based on Java components. The components can be assembled together or used separately in different application contexts. Possible applications are the constitution of electronic libraries from existing sources, the collect of information for trading systems, the assembling of fragments of data for technology watch, and more generally knowledge management and information diffusion in the enterprise. Used together, the components form a unique solution to support iterative definition and manipulation of semistructured databases derived from existing data sources or loosely structured files in Internet/Intranet environments. They are based on the new emerging standard XML and its derivative such as XML-QL, a query language for XML proposed at W3C by AT&T and INRIA.

Today Intranet/Internet platforms support access to existing files, databases, data warehouses, and legacy applications from a single browser interface. However, the existing tools do not allow the integration of multiple data sources, due to the lack of metadata generators, metadata integrators, and distributed query evaluators including update transactions. In such a context, some data sources are well structured, while others are not. The XMLMedia components provide a nice way to merge structured

and non-structured data and make it accessible through established standards, namely SQL3, JDBC, and XML, on the Internet or Intranets.

2.2 Related Work

There has been recently a lot of work on semistructured data. Some systems pioneered a storage model based on object DBMSs. TSIMMIS [15], Lore [13], and Strudel [6] store their data as graphs. [16] presents an original approach based on binary relations (one for each label) storing object identifiers and values in the context of the Monet DBMS. They use the MOA algebra to deal with type coercion and object assembling. [3] proposes an object-oriented schema derived from the DTD to store SGML documents. We further detail relational storage techniques for semistructured object in the next subsection.

Experiences in federating data sources through a semistructured model have been conducted in several projects, including TSIMMIS, Lore and YATL. This last system focuses on rule-based translation techniques and object typing [1]. For optimizing queries, several approaches have been studied. Optimization techniques are for example surveyed in [2]. More specifically, [9] optimizes regular path expressions using graph schemas.

The management of metadata is also an important problem in (federated) semistructured DBMS. Originally proposed in [10], strong dataguides can help in query formulation or in query processing [13]. Panoramic dataguide [11] were introduced for query formulation. They are quite similar to path index introduced in object systems. Recently, [12] pioneered the idea of using approximate dataguides, which group together dataguide paths when their target sets are “almost” equal.

2.3 Storing Semistructured Objects in Tables

There are obviously many ways to map graphs into tables. Hence, several mapping schemes are possible for storing semistructured objects in an object-relational DBMS. Our first experiences demonstrate that simple mapping schemes can differ a lot in performance for a given query workload according to very specific detailed choices, such as OIDs assignation, clustering and indexing schemes, number of tables to open, etc. In a first phase, we investigated the following mapping schemes:

- *The bulk data type.* A bulk data type can be used to store directly XML documents with tags and data. Access to individual structural elements is then realized by parsing. This approach is rather efficient for assembling documents, but requires specific index structures for retrieval.
- *The single-edge table.* A table can be used to represent each individual edge of the object network as a tuple, consisting of parent-OID, label, type, order, and child-OID. Conventional relational operators, including select, join, recursive join, project, and aggregate, can be used for evaluating queries over such structures.

With appropriate indices for values and links the inefficient scan can be avoided to a large extent. In contrast to using a bulk-data type, the approach is rather efficient for accessing small substructures, whereas the overhead involved in assembling complex documents could be rather high.

- *The multiple-edge table.* This scheme is a variation of the previous one where a generic table is used, but one tuple contains all edges going out from a given node. This requires nested tables. A typical scheme for the generic table consists of pairs parent-OID, array of edges, each edge being described by a label, a type, and a child-OID. A first advantage of this solution is that the order is encoded within the array; also, edges outgoing a given node are grouped together; however, searching in array-valued attributes remain a difficult operation in most object-relational DBMSs.
- *The single-edge attribute tables.* The idea is to store all the edges with the same label in a separate table having the name of the label [16]. This approach can be understood as a variation of the single-edge table but with several edge-tables, one for each edge name. A variant can be obtained by clustering or indexing the single-edge table on the label attribute. The document assembling is difficult with the single-edge attribute tables approach. However, it is efficient for selecting elements of a given label. As manipulating table names is less efficient than manipulating indexed attribute values in most object-relational DBMSs, we prefer the single-edge approach with clustering or indexing.
- *The structured tables with overflow.* An explicit relational schema can be derived from a collection of semistructured objects. The schema can be composed of several tables, each designed to group repetitive structures. STORED [5] proposes a declarative query language to specify the mapping between the semistructured data model and the relational model. The mapping exploits regular patterns that can be found in semistructured data to compose multiattribute tables. Furthermore, an overflow graph is managed to store data that do not fit the relational schema.

2.4 Querying XML documents

Several query languages have been proposed to query semistructured data and XML documents, among them LOREL, XML-QL and XQL.

The LOREL language [13] was one of the most complete query language built for querying semistructured data, more specifically OEM graphs. LOREL is basically an extension of OQL with extended path expressions. A simple path expression is a root name followed by a sequence of tags, for example `Guide.Restaurant.Adress.Street.` This expression refers to all objects reachable from the root following the edges with the given tags. Path expressions can be used to specify collections and in place of attributes for conditions and projections. They are generalized to express search patterns with string profiles, disjunction be-

tween tags, optional tags and regular expressions with the Kleen star, allowing 0 to N repetition of a tags. For example, the following request retrieve all attributes of restaurants in cities or streets containing the substring “puy” :

SELECT GUIDE.Restaurant.#
WHERE Guide.restaurant.Adresse.(Ville|Rue) **LIKE** "%puy%".

LOREL is powerful and elegant, but requires the support of OQL, which is quite complex in federated database systems.

XML-QL is another query language specifically designed for XML document collections. XML-QL is based on XML patterns, i.e., XML documents with elements replaced by variables and predicates. XML-QL provides facilities to filter document collections, but also to construct query results as new XML documents. A typical query is of the form :

WHERE <template>
CONSTRUCT <template>.

A template can be either an XML document with variables and conditions including joins, but can also be a condition with regular expressions or function calls, including Skolem functions to create new nodes. A template can also be an XML-QL query, which allows to nest queries and express powerful constructs. A simple example of an XML-QL query is :

```
WHERE
  <Guide>
        <restaurant> r
              <Address>
                    <City>  c </City>
                    <Street> s </Steeet>
                    (c LIKE “%puy%” OR s LIKE “%puy%”)
              </Address>
        </restaurant>
  </Guide>
CONSTRUCT
  <Guide> r </Guide>
```

A derivative of such languages called XQL is under a standardization by the W3C. The language is similar to XML-QL but has a different rather complex syntax. The basic idea is to extend URLs and hierarchical designators to represent queries. Special characters are introduced to encode search patterns, such as / to look for the child of a node, // to look for all descendants, * to designate any label, @ to capture an attribute, […] to nest sub-queries, ? to return a node and ?? to return a node and all its descendants. XQL could be part of the XSL standard or a stand alone component.

3 System Architecture and Components

In this section, we present an overview of the system architecture and summarize the functions of the main components.

3.1 Architecture Overview

As mentionned in introduction, the XMLMedia system federates multiple data sources using a semistructured data model and XML as the basic exchange vehicle. The system follows the DARPA I3 architecture including three layers of functions : the translation layer, the mediation layer and the coordination layer. The overall system architecture is represented in figure 1. It is composed of Java packages constituting components (they should be arranged as JavaBeans). For accessing relational DBMSs, we use JDBC. For the rest, we developed our own components.

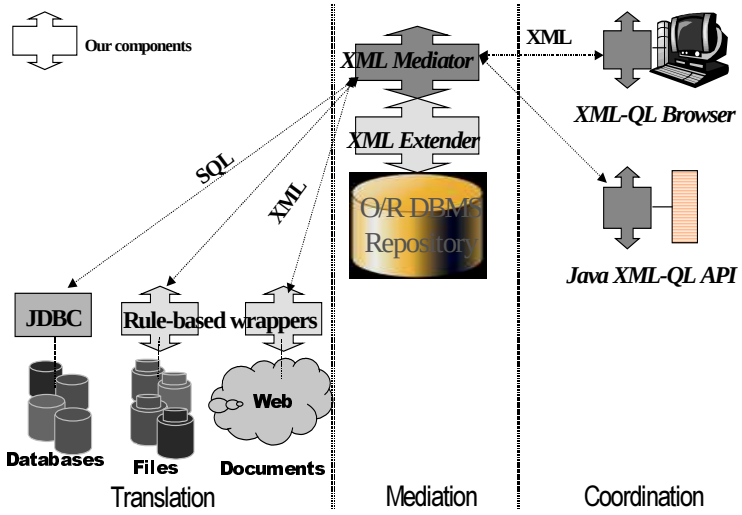

Fig. 1. System Architecture

3.2 The Three Tiers

The system follows a three-tiered architecture, with clients, application servers running the mediating components, and the data tier including most of the translation services. More precisely, we distinguish three layers as follows :

The **translation layer** includes the data sources and wrappers to wrap them in XML or SQL. XML wrappers are built from the JEDI toolkit developed at GMD in Germany [8]. JEDI is a powerful tool based on production rules. The rule allows the wrapper implementor to write search patterns in source documents and to produce variables. The variables can later be bundled in XML. Rules can call each other and can be recursively applied. Different formats of sources are supported including text, HTML, RTF, PS, DOC, XML, etc. Queries can be expressed against the source using a subset of OQL.

The **mediation layer** includes the key components we will describe in the two next sections. The XML Extender extends any JDBC compatible DBMS to support XML

document storage and querying. This component provides the necessary tools to store efficiently XML in relational tables and to run SQL3 queries extended with generalized path expressions. The query language is similar to LOREL, but built on SQL. Also, it allows result restructuring through a special AS clause. The XML Mediator does mediation between all the sources, including XML and SQL ones. It processes XML-QL queries, divides them in sub-queries according to the source capabilities, collects the results in XML, assembles them, and sends an integrated answer to the client.

The **coordination layer** includes all the client tools. It is composed of an XML-QL browser to formulate queries and present results, and of a JAVA API. This API allows Java programs to send XML-QL queries and to get the results as an XML document. The document can further be exploited in Java using the DOM tools.

4 The XML Database Extender

In this section, we describe the XML Database Extender (sometimes also called cartridge), which is able to store XML documents in any object-relational DBMS and which provides an extended SQL3 with path expressions to retrieve XML document fragments and reassemble them.

4.1 XML Document Modeling

As in many research projects, we use labeled directed graph for modeling semistructured object. Our modeling is an ordered version of the OEM model [13] extended with different edge types. A document consists of a collection of objects, in which each object is either complex or atomic. A complex object is an optionally ordered set of <name, object> pair. An atomic object is a value of type int, real, string, image, video, and more generally of any type supported by the underlying object-relational DBMS (user data types are possible). The optional order gives the relative order of edges among those going out from a given node. To better model related XML documents, we distinguish two types of edges, as shown in Fig. 1: an edge originated and targeted within the same document is called an *aggregation link*, and an edge between different documents is called an *association link*. Hence, a document is a graph, with edges labeled by names and leaves labeled by values. Documents can be linked together by association links.

When assembling a document, only links of aggregation type are considered as relevant. This information is particularly important to model XML documents, which includes M to N relationships (i.e., association links). The order is simply memorized through an ordinal number. This is important to assemble XML document items in the correct order. Figure 2 shows on an example how we model XML linked documents.

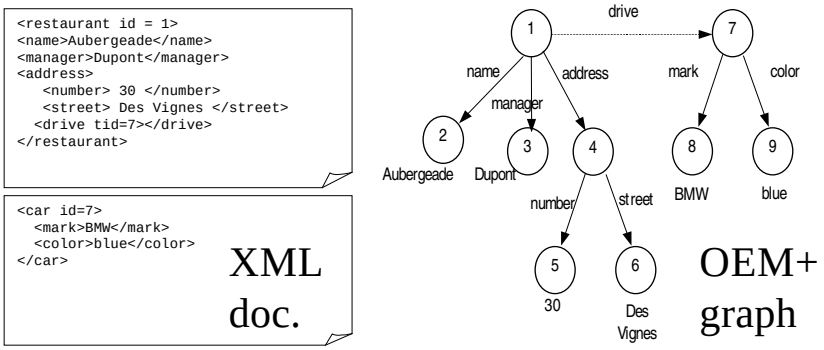

Fig. 2. XML document modeling

4.2 Storing Documents in Tables

For simplicity and efficiency in retrieval, we selected the single-edge table approach. Edges in the graph are stored in a table called the *edge table***.** In the database, each node is identified by a unique identifier. For allowing efficient clustering of the document edges, the root node identifier of each document is repeated as the first attribute of each edge of the document. Hence, each edge is represented by a flat tuple [root, origin, target, label, link type, node type, order]. The root attribute indicates the object root OID from which the edge is a descendent. The origin and target attributes memorizes the node OID of the origin and target node respectively. Label is the label of the link. In the edge table both *aggregation* and *association* links are stored. Hence, link type indicates whether a link is a composite aggregation or an association. Node type is the type of the value attached to the target node. For internal nodes, it is simply "NODE", while it is the name of the leaf table containing the value for a leaf node. The structure of the edge table is exemplified in Figure 3. To group edges of a document in one or more pages, the *edge table* is clustered on root_oid. A clustered index is maintained by the RDBMS to give direct access to edges of a document, e.g., to join efficiently with a relational table.

root	origin	target	label	ltype	ntype
1	1	2	name	AGGR	STRING
1	1	3	manager	AGGR	INTEGER
1	1	4	address	AGGR	NODE
1	4	5	number	AGGR	INTEGER
1	4	6	street	AGGR	STRING
1	1	7	drive	ASSC	NODE
7	7	8	mark	AGGR	STRING
7	7	9	color	AGGR	STRING

Fig. 3. An instance of the edge table

Leaf nodes with their values are stored in *leaf tables*. By default, one leaf table is used to store leaves of each type. Special mapping specifications can be used to define target tables for several leaf nodes. For the time being, the mapping language allow the administrator to stipulate which XML element is going to be stored in a table. The element has to be a tuple conforming to the value table schema. We envision to extend the mapping language for mapping complex values in a unique object-relational table in the future.

As mentioned above, this storage structure is quite efficient for query processing. Indices can be declared on origin and taget OIDs and labels, to accelerate selection and join when querying. The problem with this approach is the cost of assembling full documents data from the leaf tables. As tuples are randomly distributed within the leaf table, assembling a complex document may require several I/Os in the leaf table. To reduce the high cost of assembling document values, we use a clever node OID allocation policy. A unique integer value is assigned to each node as its identifier using a sequence number (provided by the RDBMS). Within an XML document, ascending node OIDs are assigned to each node in the breath-first traversal order. Thus, identifiers of leave nodes of a document are close together. We cluster leave tables on their identifiers and index them on that identifier. To retrieve the data of a document, range queries can be used in each leaf table containing data from that document. Figure 4 illustrates a simple leaf table for storing the string data type values. Notes that leaf tables are clustered on NODE ID, to group chunks of documents together.

NodeID	Value
3	“Dupont”
6	“des Vignes”
8	“BMW”
9	“blue”

Fig. 4. Example of a leaf table for string values

4.3 Document Query Processing

As already mentioned, we use an extended SQL to query XML documents. The extension for SQL is similar to LOREL for OQL, i.e., we mainly support generalized path

expressions in SQL queries. Generally, relational systems are considered inefficient for processing queries containing path traversals. However, in the context of semistructured data, some characteristics of relational systems can compensate the handicap. First, in object-relational systems, path traversals can be processed in a set-oriented manner using successive joins on indexed attributes (the source and target OIDs). Second, in systems supporting recursive queries, path expressions can be processed using them. For example, in Oracle 8, the hierarchical recursive query is well suited for evaluating path expressions.

Obviously, the cost of path traversals increases as the path length increases. To optimize long path traversals, our implementation support the concept of *path index*. A path index is simply a join index of the edge table with itself, using target equals origin as join predicate. The path name is kept by concatenating the edge names. By joining again the path index with the edge table, we can generate path index of length 3, and so on up to path index of length n. Path index tables can become large. To reduce the size of path index tables, we can choose to store only paths frequently occurring in the graph. A no match will then requires normal processing against the full edge table. A path index reduces the number of joins or level of hierarchical traversal in processing path expressions. To traverse a given path expression, we first search within the path index corresponding to the maximum covered path length, and then follow up within the edge table. A path index of length *n* cannot be used directly to evaluate path expressions of length shorter than *n*.

In semistructured databases, regular path expressions with Kleene stars have special importance. Semistructured data have irregular and changing structures. By means of regular path expressions, users can query the database without the full knowledge on data structures. Although convenient to use, regular path expressions in general cannot be evaluated directly due to their prohibitive cost. Up to now, the proposed solutions for optimizing regular path expressions are all based on pruning techniques [9]. To apply pruning, some sort of metadata is required. Several variations of the representative objects [14] and dataguides [10] including the approximate dataguides [12] have been proposed. They are useful to keep track of the possible nesting of labels and avoid full scan of the database.

In our system, any label can be accessed directly regardless the distance from the root. This is due to the existence of an index on the labels. In addition, the graphs can be traversed in forward or reverse direction through joins. The traversal can be efficient if path index are maintained, as explained above. These features are quite helpful for processing path expressions with wildcard and more generally Kleene stars.

To further accelerate path expression processing, we maintain a specific sort of metadata called the *longest path set*. Informally, this is the set of longest paths in the graphs. Queries based on path expressions search for nodes but are formulated from labels. Hence, in our structure, we keep only labels. We model the possible paths as a regular language where labels are letters from an alphabet, and where regular path expressions are words (infinite words in the event of loops in the graph). The goal being to determine words that belong to this language, we do not keep words that are factors of a longer word. Regular path expressions have the following format: $l_0.l_1...l_i.*.l_j.l_{j+1}...l_n$. where the Kleene star matches any path of any length. The goal of

the rewriting is to find all sub-words from the set of words in the longest path set which start with $l_0.l_1...l_i$ and finish with $l_j.l_{j+1}...l_n$. Then, the query can be transformed in a query with a disjunction of fully documented path expressions in qualification.

In summary, we expand regular path expressions using the longest path set words and sub-words. For simplicity, we assume that graphs have no loop. It is possible to handle loops, but they have to be detected at document loading and expanded with special labels to encode infinite longest paths.

5 The XML Mediator

The XML mediator is the other key component of the mediation layer. Through the browser, the client issues an XML-QL query to the mediator and waits for an answer. As usual in mediation systems, the mediator accepts the query and decomposes it into subqueries, one for each source, with in addition a recomposition subquery. Subqueries are routed to a gateway in charge of shipping the query to the appropriate wrapper. The gateway is able of managing a pool of data source connections to save overhead. When the connection has been allocated, the gateway issues the subqueries to the wrappers and waits for a response. The wrappers process the subqueries by consulting the associated data sources and generate subanswers that are returned to the mediator. The mediator combines the subanswers by using the composition subquery and generates the final answer that is returned to the client

In addition, the mediator provides some facilities to translate queries according to the capabilities of the wrapper. To perform this translation, specialized components called *doors* are included. A door is a generic class and can be extended by query and result translation methods. Figure 5 gives an overview of the mediator architecture.

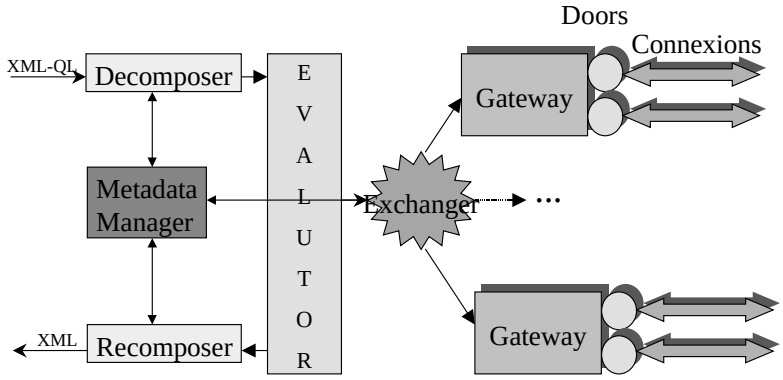

Fig. 5. The mediator architecture

The role of the main components of the mediator can be summarized as follows :

- The Decomposer decomposes XML-QL queries in query graphs.

- The Evaluator identifies local subqueries to ship and run the recomposition query.
- The Exchanger routes the query to the appropriate gateway and collects the result back.
- The Gateway manages connection pools to data sources and submits subqueries to wrappers.
- The Door translates queries for local wrappers and translates answer back in OEM graphs
- The Recomposer recomposes an XML answer for the client.
- The Metadata manager collects and maintains schemas and label paths for each data source.

In summary, the XML mediator is somehow similar to a relational mediator except that it is based on XML-QL and returns data in XML. The semi-structured OEM+ model supported by data and query graphs is the basis of our internal implementation. Thus, XML data are internally represented as extended OEM graphs similar to that manipulated in the XML extender described in the previous section. Queries are also represented as graphs with constraints (i.e., restrictions) and links (i.e., joins). Extended OEM graphs support easily the mixing of relational data and true semi-structured data provided by XML wrappers or SGML databases.

6 Conclusion

In this paper, we have presented XMLMedia, a mediating system based on XML. XMLMedia federates both XML documents and SQL tables and unify them in a virtual XML digital library. The XML Extender provides the necessary functionnalties to store and manipulate XML documents on top of an object-relational DBMS. Coupling the capabilities of the extender with multimedia features now available in most object-relational DBMSs gives powerful functions to retrieve documents by contents, e.g., using similarity search on keywords or image features. The mediator offers the XML-QL query language to query the virtual libraries of XML documents.

A first version of the system is already experimented in several applications as a test-bed, including a tourism virtual Web site in Tyrol, an Intranet in a Spanish hospital and a trading application. Performance is achieved by the nice features well established of object-relational systems and by appropriate pools of buffers and connections. The mediator can also be used to feed Web data in a datawarehouse and more generally to collect and disseminate information in the enterprise. A demonstration of the system can be found on the Web at http://nora.prism.uvsq.fr/miroweb/.

Our experience in federating data sources through XML and XML-QL demonstrates the advantages of the semistructured data model for exchanging data. First, it is flexible and auto-documented. No type requires to be predefined, as with the object model for example. Second, it is easy to merge semistructured data and relational one through direct mapping, which can be more or less sophisticated for optimization

purpose. Third, XML is an appropriate exchange model on the Web and writing wrappers is an easy task, especially with a powerful wrapper generator as is JEDI. Fourth, XML-QL is appropriate for client-server dialog, as it is XML uniform (a query can be seen as an XML document). Fifth and it is probably the most important, XML gives good opportunities to unify classical database processing with information retrieval approaches. The power of the resulting tool is not well evaluated yet.

Acknowledgments

We would like to thank Peter Fankhauser, Dana Florescu, Henri Laude and Patrick Valduriez for helpful discussions, particularly in the context of the MIRO-Web project. We owe them many ideas.

References

1. Serge Abiteboul, "*Querying Semistructured Data*", in Proceedings of the 6th International Conference on Database Theory, Delphi, Greece, 1997
2. Peter Buneman, "*Semistructured Data*", in Proceedings of the 16th Symposium on Principles of Database Systems, Tucson, Arizona, 1997
3. Vasilies Christophides, Serge Abiteboul, Sophie Cluet, Michel Scholl, "*From Structured Documents to Novel Query Facilities*", in Proceedings of the 1994 ACM SIGMOD International Conference on Management of Data, Minneapolis, USA, May 1994
4. Alin Deutsch, Mary Fernandez, Dana Florescu, Alon Levy, Dan Suciu, "*XML-QL: A Query Language for XML*", 1998, http://www.w3.org/TR/NOTE-xml-ql/
5. Alin Deutsch, Mary Fernandez, Dan Suciu, "*Storing Semistructured Data with Stored*", ACM SIGMOD International Conference on Management of Data, SIGMOD Record Vol. 28, N° 2, Philadelphia, June 1999.
6. Peter Fankhauser, Gerald Huck, "Cooking XML : An Extraction Tutorial for Jedi, GMD, Darmstadt, April 199, http://www.darmstadt.gmd.de/~huck/jedi/tutorial
7. Peter Fankhauser, Georges Gardarin, Moricio Lopez, J. Muntz and Antony Tomasic, "*Experiences in Federated Databases: From IRO-DB to MIRO-Web*", in Proceedings of the 24th International Conference on Very Large Data Bases, New York, USA, August 1998
8. Mary Fernandez, Daniela Florescu, Alon Levy and Dan Suciu, "*A Query Language for a Web Site Management System*", SIGMOD Record, vol. 26, no. 3, pp. 4-11, 1997
9. Mary Fernandez and Dan Suciu, "*Optimizing Regular Path Expressions Using Graph Schemas*", in Proceedings of the 14th International Conference on Data Engineering, Orlando, Florida, USA, February 1998
10. Roy Goldman and Jennifer Widom, "*DataGuides: Enabling Query Formulation and Optimization in Semistructured Databases*", in Proceedings of the 23rd International Conference on Very Large Data bases, Athens, Greece, 1997
11. Roy Goldman, Jennifer Widom, "*Interactive Query and search in Semistructured Databases*", International Workshop on the Web and Databases, Valencia, Spain, March, 1998.

12. Roy Goldman, Jennifer Widom, *"Approximate Dataguides"*, International Workshop on Query Processing for Semistructured Data and Non-Standard Data, Jerusalem, Israel, January, 1999.
13. Jason McHugh, Serge Abiteboul, Roy Goldman, Dallan Quass, Jennifer Widom, *"Lore: A Database Management System for Semistructured Data"*, SIGMOD Record, 26(3): 54-66, September 1997
14. Svetlozar Nestorov, Jeffrey D. Ullman, Janet L. Wiener, and Sudarshan S. Chawathe, "*Representative Objects: Concise Representations of Semistructured, Hierarchial Data"*, in Proceedings of the 6th International Conference on Database Theory, Delphi, Greece, 1997
15. Yannis Papakonstantinou, Hector Garcia-Molina, Jennifer Widom, *"Object Exchange Across Heterogeneous Information Sources"*, in Proceedings of the 11th International Conference on Data Engineering, Taipei, Taiwan, 1995
16. Roelof van Zwol, Peter Apers, Annita Wilschut, *"Implementing Semi Structured Data with MOA"*, International Workshop on Query Processing for Semistructured Data and Non-Standard Data, Jerusalem, Israel, January, 1999.