Integrating Heterogeneous Data Sources with XML and XQuery

G. Gardarin, A. Mensch, T. Tuyet Dang-Ngoc, L. Smit
e-XMLMedia, 31 Avenue du Général Leclerc
92340 BOURG LA REINE, France
georges.gardarin@e-xmlmedia.fr

Abstract

XML has emerged as the leading language for representing and exchanging data not only on the Web, but also in general in the enterprise. XQuery is emerging as the standard query language for XML. Thus, tools are required to mediate between XML queries and heterogeneous data sources to integrate data in XML. This paper presents the e-XMLMedia mediator, a unique tool for integrating and querying disparate heterogeneous information as unified XML views. It describes the mediator architecture and focuses on the unique distributed query processing technology implemented in this component. Further, we evoke the various applications that are currently being experimented with the e-XMLMedia Mediator.

1. Introduction

In recent years, there have been many research projects focusing on heterogeneous information integration. Typical information integration systems have adopted a wrapper-mediator architecture [1]. In this architecture, mediators provide a uniform user interface to query integrated views of heterogeneous information sources. Wrappers provide local views of data sources in a uniform data model. The local views can be queried in a limited way according to wrapper capabilities. Well-known research projects and prototypes based on this architecture include Garlic [2], Tsimmis [3], IRO-DB [4] and Yat [5]. While in the 90's most studies were based on using the object model as data integration model, the focus has come to XML as pivot model at the beginning of the new century.

The advantages of XML as an exchange model, (i.e., it is rich, clear, extensible and secure), makes it the best candidate for supporting the integrated data model. In addition, using XML views for local data sources hides the local specificities of each system. Furthermore, the richness of the XML schema model simplifies wrapper mappings. Also, the emergence of XQuery as a powerful universal query language for XML makes it possible to query XML global and local views in a uniform way based on a standard interface. Thus, these advantages explain that several research projects have emerged to query in a uniform way heterogeneous data sources based on XML as exchange model, see for example [6, 7, 8].

e-XMLMedia is providing one of the first products based on XML to integrate heterogeneous data sources, namely the e-XML Mediator. The mediator (with the associated wrappers) provides the required functionalities to query in a uniform way heterogeneous data sources. It is a sophisticated component composed of several packages in charge of decomposing queries into mono-source sub-queries, efficiently shipping local sub-queries to

data sources, getting results in XML through a SAX interface, processing and assembling them. Queries as well as sub-queries are expressed in XQuery. In addition, capabilities are associated to wrapper so that the mediator sends only supported queries to wrappers. In summary, the mediator uses XML to represent disparate data in a common format and create a unified view of that data. Using advanced distributed query processing technology, the mediator provides an application with the services it needs to integrate on demand heterogeneous information.

This paper gives an overview of the e-XML Mediator. The next section focuses on the middleware objectives. Then, we briefly describe the system architecture. In section 4, we give an overview of the unique query processing technology embedded in the component. In conclusion, we focus on typical applications for the mediator.

2. Mediator Functional Objectives

The mediator is a data integration middleware receiving XQueries from user applications and returning XML. Internally, XML is managed as event flows using SAX. In an effort to broaden the interoperability of products in this market segment, e-XMLMedia has defined a standard XML XQuery API for the access to XML based data products, called XML/DBC. The XML/DBC API consists of two closely related APIs. The Java API is a JDBC extension to query XML collections using XQuery. The Web Services API is based on SOAP and is designed to provide a SOAP style server interface to clients. Thus, the mediator as well as other e-XMLMedia products, receives queries through its XML/DBC API either from SOAP client or from Java clients.

The mediator objectives are summarized in the following sections.

2.1. Querying heterogeneous data sources

The mediator provides views of sources containing collections of XML documents that can be queried through XQuery, the W3C emerging standard. It takes into account the capabilities of the data source, to avoid sending not executable sub-queries to a source. Each data source is encapsulated within a wrapper that supports at least a subset of XQuery. Ideally, a wrapper can provide mapping functionalities as XML views to achieve local mappings of data and metadata.

2.2. Querying distributed data sources

The data sources can be distributed on various sites interconnected through intranets or the Internet. Various interconnection protocols can be supported, including HTTP, IIOP and, at a higher level, RMI and SOAP. A mediator can appear itself as a data source, thus allowing several layers of mediation.

2.3. Composing data from distributed sources

Each wrapper and the mediator itself support XQuery and provide XML results. The data from the sources can be combined using various operations such as union, cross product, join, fusion, reordering, nesting and arranged to comply with the requested output defined in the return clause of the XQuery.

2.4. Processing queries in an efficient way

Queries are decomposed in optimal mono-source sub-queries and global query plans are optimized in a simple but efficient way. Simple heuristics are supported in a version 1, while cost-based query optimization should be introduced in the future. Heuristics include the

XML counter-part of classical relational detachment of selections and semi-join transformations.

2.5. Efficiently caching metadata

To discover relevant sites for a query and decompose it, metadata are maintained describing the sources. At mediator start, only first level metadata are brought in main memory. These metadata are given to the mediator through a configuration file. Further metadata are searched on demand through a wrapper interface and kept in cache on a last recently used basis.

2.6. Supporting efficiently multiple users

Multiple users are able to process queries in parallel with little overhead due to query processing sharing and multi-threading. The mediator is multi-threaded and manages multiple connections to data sources.

3. Mediator architecture

The mediator architecture is represented in Figure 1. The XML/DBC API is the only interface with external components. Thus, notice that the mediator ships requests to wrappers through XML/DBC and thus get results through it. This makes possible for a mediator to see another mediator as a wrapper. Futhermore, results are supplied in XML/DBC through SAX readers. Thus, flows of events are transferred between mediators and wrappers, avoiding the overhead generated by the allocation of intermediate memory structures.

The major sub-components are the XQuery parser, the metadata manager, the query evaluator, the query decomposer, and the result reconstructor. They are briefly described below.

3.1. XML/DBC API

This is a common API for both the mediator and the wrapper to receive external queries and ship XML results. This interface provides functions to manage components (start and stop a connection) and to retrieve data or meta-data.

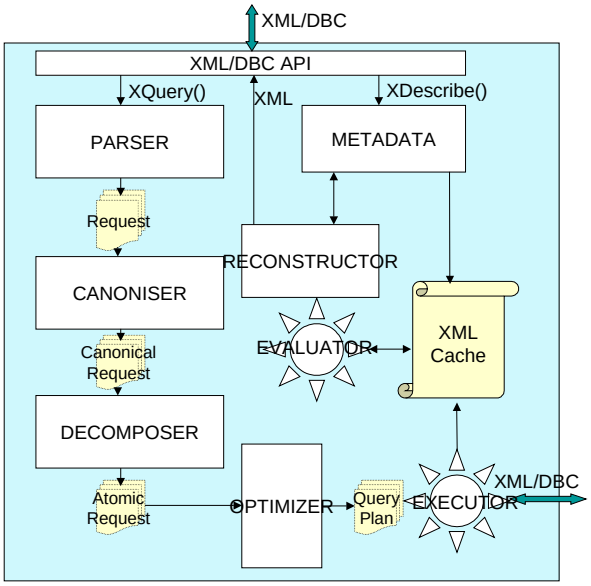

Figure 1 – Overview of the mediator architecture

3.2. Parser

The parser parses the query and generates the query structure if the query is syntactically correct. Otherwise, it returns a documented error.

3.3. Canoniser

The canoniser normalizes the query and generates a query in canonical form. Canonization consists in extracting simple

queries from the external query and in building the encapsulating global query, which is composed of a tagging operator and may include specific nesting and aggregation operators.

3.4. Decomposer

The decomposer decomposes each simple query in atomic queries, according to the data localization, which is extracted from the meta-data. In general, the decomposition moves further operators in the global query.

3.5. Optimizer

The optimizer identifies the relevant data sources based on metadata and create an execution plan for the query. The execution plan is composed of operators of a specific algebra designed for XML. Simple optimizations of the query plan are performed in the current version, but more complex ones are planned based on a cost model. For example, the optimizer groups the operators which refers the same source in a single query for shipping once. It also orders the global operators according to query heuristics and finally selects the best processing method (parallel, sequence or pipeline) for global operators.

3.6. Executor

The executor is in charge of shipping the sub-queries to the wrappers using XML/DBC and collecting the results in cache memory. In general, results are not fully instantiated in main memory but SAX events are produced and directly processed by the evaluator when possible. We represent each ordered collection of XML tree shipped from a wrapper as an Xtuple, i.e., a tuple of references to forest of XML trees instanciated in cache.

3.7. Evaluator

Based on the query plan, the evaluator evaluates the remaining global query and applies the algebraic operators in main memory. We use a specific algebra designed for processing Xtuple. The operators are able to perform XPath-based projection, restriction, product, join, grouping, sorting, union, intersection and difference of ordered collections of Xtuples. For each operator, we implement one or more specific algorithms. For example, several global join algorithms are possible. The evaluator may work with intermediate collections fully stored in main memory, but can also work on a SAX flow of events, thus implementing pipelining and hash joins. Dependent join algorithms requesting Xtuple to one source and querying the other based on the results are also possible.

3.8. Reconstructor

It applies the result generation part of the global query to the intermediate results generated by the evaluator and the sources to construct the final result. Finally it tags the results in the required way and built the SAX event flow.

3.9. Metadata manager

This package manages the schemas of all registered sources. Further, for each source, it maintains the collection names with the associated queryable path set. The path set is a kind of data-guide giving an overview of all path instantiated in the source. If a path is not given, it will not be queried. The path set has to be given by the wrapper when registering the source (on command XDescribe).

4. Query processing

Globally, queries are normalized, i.e., transformation rules are applied to eliminate nested queries and replace them by several mono-bloc queries. Next, canonization is applied, which transforms a query in sub-queries nested in a reconstruction operator. Each sub-query is in turn divided in atomic query, one for each relevant source. Local sub-queries are processed in one or multiple shots according to wrapper capabilities and shipped to local wrappers. Results are then post-processed to compute the global answer based on our XML algebra. Finally, a reconstruction process applying a tagging operator completes the query processing steps.

4.1. Canonical queries

There exist several forms of queries. External queries are any queries that can be issued by a user. Simple queries are the XML counterpart of SPJ queries in relational systems, i.e., queries performing selections and joins over collections of XML trees, but returning tuples as results. They are of the form:

for X1 **in** P1, X2 **in** P2, … Xn **in** Pn
where Condition(X1, X2, …Xn)
return Tuple(X1, X2, …Xn)

where the Xi's are variables, the Pi's are simple path expressions, Condition a logical expression constructed from the Xi's and usual comparison operators, Tuple a simple constructor of Xi's tuples. The result of such a query can be stored in a temporary table of schema [X1, X2, …Xn]. Note that depending on the data sources and path expressions, the Xi's can range in complexity from atomic values to complex XML sub-trees.

Canonical queries are more complex and have the same selection power of external queries, but they are arranged as flat queries without imbrications giving Xtuples as results. They are more complex than simple queries as they can include aggregates and functions. They are nested in a global query that performs result reconstructions. For distributed queries, the global query may also include distributed joins and aggregate computations and in general algebraic operators. After completing the distributed processing, the global query processes the results of the nested queries to perform result restructuring as derived from the XQuery return clause.

4.2. Query rewriting

Atomic queries are queries involving a single collection. Notice that atomic queries may refer to several XPath expressions on the same collection. Atomic queries are simple queries, but simple queries are in general not atomic, as they may involve several collections. They are introduced for processing localization of data.

Internally, two data structures are manipulated for representing queries: The request structure is the result of parsing. It is a parse tree representing the query, which is supported by a complex main memory structure. A query plan is a tree of algebraic operators representing the query to be shipped to local sources and a global query to evaluate locally. The query plan could be enriched in the future with annotations of operators, at least at the level of the global query, e.g., to indicate pipelined hash-join, dependant join, etc.

4.3. Query plan optimization

The query processing component is composed of several packages in charge of decomposing XQuery queries into mono-source

sub-queries, efficiently shipping local queries to data sources, gathering results, processing and assembling them. In particular this component chooses an execution strategy for the query.

The Mediator makes three key decisions during query processing. First the Mediator determines the localization of the relevant sources to process a query among those accessible by the Mediator. Using the metadata information on each data source, the location of data that appears in a query is determined. Second the Mediator determines the amount of sequential or parallel execution that occurs during execution. The mix between parallel and sequential is determined by the binding relationship between different data sources as expressed by join predicates. Independent data sources are contacted in parallel. Dependent data sources are accessed in a pipelined fashion. Third, the Mediator determines the amount of query processing that can be offloaded onto the data source. This determination is the result of a comparison of the requirements of a particular query and the query processing power available to a data source.

Further, the mediator creates query plans that are trees of XML algebra operators. It also executes them. As explained above, these operators process collections of Xtuples. They can be re-ordered in query plans according to equivalence rules derived from the relational algebra and from specific navigation rules. Each operator can have several implementations. Thus, generating the best query plan for a request is a complex problem. A cost-based solution has been designed but only simple heuristics are implemented so far.

The Mediator also provides a way to structure the result in an integrated XML document, according to the RETURN clause of the query. The three main facilities provided at this level are the following:

- Renaming Tags;
- Changing the initial data structure;
- Composing results when coming from multiple data sources.

4.4. Wrapping Data Sources

Data sources are managed through an XML based configuration file. Data sources are wrapped with a standard interface. They are distinguished as wrapped relational data sources, wrapped XML data sources not supporting complex queries other than selections, and XML Repositories supporting XQuery via the XML/DBC interface. Wrappers are available for SQL (object-relational DBMS accessed through JDBC), XQuery (Repository and Mediator), and HTML pages. Specific wrappers can be added; they have to comply with the XML/DBC interface and define their capabilities as a class.

4.5. Metadata management

If the Mediator did not have knowledge about where to find data relevant to a given query, it would have to scan all the registered data sources. This would not be acceptable in terms of performance, as this scan should have to be done each time a query is processed. The solution to this problem is to maintain metadata describing the sources. When a source is registered with a mediator, the mediator asks for the source metadata. The metadata must be passed to the mediator as an XML document containing the source schema. Metadata are used to determine the location of data on the sites, both to identify and ship every subpart of the request to the appropriate gateway. They are also used to retrieve for the user the metadata structure available for requested objects. If metadata are modified in a source, the metadata associated with the source must be refreshed.

5. Typical applications

Several applications are currently built using the e-XML Mediator. A simple kind of application is the publishing of relational data as integrated data in XML. We packaged our XML/DBC wrapper for object-relational databases as a component marketed under the name XMLizer, to transform any relational source in an XML data source supporting XQuery. For example, the XMLizer is a key component in several database interchange, XML EDI and XML portal applications. More complex applications using the Mediator include portals for querying multiple heterogeneous databases. We have also developed applications in cooperation with European partners for a tourism Web site federating multiple data sources, for a virtual hospital federating patient dossiers constituted from several pieces, and for an active document publisher composing documents from several sources including databases and reports. In general, the mediator is ideal for extracting and composing disparate information as unique XML documents. Coupled with the other products of e-XMLMedia, XML Repository and XForms Engine (XFE), the Mediator and XMLizer are ideal to develop gateways between existing information systems and new XML consuming applications.

6. References

[1] Wiederhold G.: "Intelligent Integration of Information", ACM SIGMOD Conf. On Management of data, pp. 434-437, Washington D.C., USA, May 1993.

[2] Haas L., Kossman D., Wimmers E., Yang J.: "Optimizing Queries across Diverse Data Sources", 23rd Very Large Data Bases, August 1998, Athens, Greece, 1997.

[3] Chawathe S., Garcia-Molina H., Hammer J., Ireland K., Papakonstantinou Y., Ullman J., and Widom J.: "The TSIMMIS Project : Integration of Heterogeneous Information Sources", IPSJ Conference, pp. 7-18, Tokyo, Japan, October 1994.

[4] Fankhauser P., Gardarin G., Lopez M., Muñoz J., Tomasic A.: "Experiences in Federated Databases: From IRO-DB to MIRO-Web", 24rd Very Large Data Bases, pp. 655-658, August 24-27, 1998, New York City, New York, USA, 1998

[5] Cluet S., Delobel C., Siméon J., Smaga K.: "Your Mediators Need Data Conversion", ACM SIGMOD Intl. Conf. on Management of Data, pp. 177-188, Seattle, Washington, USA, 1998.

[6] Christophides V., Cluet S., Siméon J.: "On Wrapping Query Languages and Efficient XML Integration", ACM SIGMOD 2000, pp. 141-152, May 16-18, 2000, Dallas, Texas, USA. SIGMOD Record 29(2) ACM 2000.

[7] Manolescu I., Florescu D., Kossmann D.: "Answering XML Queries over Heterogeneous Data Sources", 27th Very Large Data Bases, pp. 241-250, Roma, Italy, Sept. 2001.

[8] Shanmugasundaram J., Kiernan J., Shekita E., Fan C., Funderburk J.: "Querying XML Views of Relational Data", Proc. Of the 27th International Conference on Very Large Data Bases, pp. 261-270, Roma, Ital., Sept. 2001.