

A Query Algebra for XML P2P Databases

Carlo Sartiani

Dipartimento di Informatica - Università di Pisa
Via F. Buonarroti 2 - 56127 - Pisa - Italy

sartiani@di.unipi.it

ABSTRACT

This paper describes a query algebra for queries over XML p2p databases that provides explicit mechanisms for modeling data dissemination, replication constraints, and for capturing the transient nature of data and replicas.

Categories and Subject Descriptors: H.3.5[Information Storage and Retrieval]: Online Information Service - Data sharing; H.2.5[Database Management]: Heterogeneous Databases; H.2.3[Database Management]: Languages - Data description languages (DDL)

General Terms: Algorithms, Management, Languages

Keywords: Query algebras, peer data management systems, XML

1. INTRODUCTION

In the *peer-to-peer* (p2p) model, systems are composed of an *open-ended* and dynamic network of peers, which share data, computational resources, etc; peers are usually autonomous or semi-autonomous, and may cooperate together in the execution of tasks or in the hosting and querying of data. In the field of database research, p2p systems affirmed as an interesting evolution of past distributed and integration systems. Several *ongoing* projects focus on the design of p2p database systems, mostly for XML data. One missing point in the current research about p2p XML databases is the definition of a proper query algebra. Existing query algebras, most notably the *official* algebra by W3C [2], have been defined in the context of static and centralized database systems, and cover issues ranging from query result analysis and query type-checking to the rigorous definition of the static and dynamic semantics of XML query languages. As a consequence, they lack support for key issues in p2p database systems, such as: the dissemination of data over multiple peers, which may appear and disappear unpredictably; the replication of data in multiple copies, which have a limited time validity; and the dynamic nature of data distribution and replication.

This paper shortly describes a query algebra for XML p2p databases. The proposed algebra supports the FLWR core of XQuery, and, unlike existing query algebras for XML data, provides explicit mechanisms for modeling data dissemination and replication constraints: in particular, the algebra data model incorporates the notion of *locations*, which model peer content, as well as the notion of data freshness; moreover, the algebra provides operators for manipulating locations, and for expressing replication constraints.

Copyright is held by the author/owner(s).

WWW2004, May 17–22, 2004, New York, New York, USA.
ACM 1-58113-912-8/04/0005.

2. DATA MODEL AND TERM LANGUAGE

Data in the system are represented as unordered forests of node-labeled trees. According to the term grammar shown below, each node (n) has a unique *object identifier* (oid) that can be accessed by the special-purpose function *oid*, and is augmented with the indication of the hosting peer (*location* in the following) as well as with a freshness parameter *fr*, which indicates when the last update on the node was performed (\perp indicates that the freshness is undefined).

$$\begin{array}{llll} t ::= t_1, \dots, t_n & | & n[t] & | & n & \text{trees} \\ n ::= (oid, loc, fr)label & & & & & \text{nodes} \\ loc : (dbname \rightarrow t, (dbname, loc) \rightarrow t) & & & & & \text{locations} \\ \text{where } label \in \Sigma^*, fr \in \mathbb{N} \cup \{\perp\}, \text{ and} & & & & & \\ loc^1 \text{ and } loc^2 \text{ are partial functions.} & & & & & \end{array}$$

Locations model the content of peers, hence they are represented as a pair of partial functions: the first function (loc^1) returns, for each database identifier, the trees contributed to the database by the given peer, if any; the second function (loc^2), instead, describes the replication services offered by a given peer, i.e., it returns, for each database identifier and location, the replicated trees for such database and location, if any. Replicas are further described by a (distributed) set *replicas*, which contains dynamic replication constraints. A replication constraint has the form $(loc_1, loc_2, db, \delta_1, \delta_2)$, and it states that loc_2 replicates the content of loc_1 for the database db from time δ_1 to time δ_2 (δ_2 may assume the special value ∞ , which indicates that the replica is always kept up to date); given the dynamic nature of the system, we expect replication constraints to evolve over time.

The set of locations containing data relevant for a given database db is returned by the function *Locs*. We expect *Locs* to be computed during the generation of the query plan; as usual in p2p systems, the computed set *Locs* will be a subset of the set of all relevant locations, or, even worst, a partially overlapping set.

3. ALGEBRA OPERATORS

The algebra exploits *relational-like* intermediate structures, called *Env*, to accumulate variable bindings collected during query evaluation. *Env* structures, represented as node-labeled trees conforming to the algebra data model, are manipulated by quite traditional operators, such as *Selection*, *Projection*, *TupleJoin*, *DJoin*, *Map*, and *GroupBy*. In addition to these operators, the algebra features *location* operators, used for manipulating locations, and *border* operators, used for performing conversions from data model instances to *Env* structures, and *vice versa*.

To support dynamic replication constraints, we assume that each query has two time parameters: the query issuing time τ' , and the maximum replica time $\delta_{\tau'}$, which indicates that replication constraints of the form $(loc_1, loc_2, db, \delta_1, \delta_2)$ with $\delta_2 \geq \tau' - \delta_{\tau'}$ can be considered during query plan generation.

3.1 path and return

The main task of the *path* operator is to extract information from the database, and to build variable bindings. The way information is extracted is described by an *input filter*; a filter is a tree, describing the paths to follow into the database (and the way to traverse these paths), the variables to bind and the binding style, as well as the way to combine results coming from different paths.

While the *path* operator extracts information from existing XML documents, the *return* operator uses the variable bindings of an *Env* to produce new XML documents. *return* takes as input an *Env* structure and an *output filter*, i.e., a skeleton of the XML document being produced, and returns a data model instance (i.e., a well-formed XML document) conforming to the filter. This instance is built up by filling the XML skeleton with variable values taken from the *Env* structure: this substitution is performed once per each tuple contained in the *Env*, hence producing one skeleton instance per tuple.

3.2 Operators on Locations

Operators on locations are crucial for retrieving data coming from multiple peers, and for exploiting, if necessary, replicas of the content of some location. The query algebra offers two location operators: *LocUnion*, and *Choice*.

LocUnion (\bullet) takes as input two locations loc_1 and loc_2 , and it returns a new location obtained by uniting the content and the replica functions of the arguments. *LocUnion* is primarily used for expressing queries retrieving data from multiple peers, as shown by the following Example.

EXAMPLE 1. Consider a real-estate market database, and assume that locations loc_1 , loc_{11} , loc_{13} , and loc_{17} share data about buildings. Suppose that you want to retrieve the price and the description of each building in the database, as shown below:

```
for $b in input()//building,
    $d in $b/desc,
    $p in $b/price
return <entry> {$d, $p} </entry>
```

This query can be expressed by the following algebraic expression:

$$return_{entry[\nu \$d, \nu \$p]}(path_{(//, \$b, in)building[(/, \$d, in)desc[0]]}((\bullet_{i=1,11,13,17}loc_i)^1(db1)))$$

The *Choice* ($|_{db}^\delta$) operator is a convenient way to encapsulate replication constraints into query plans. $loc_1 |_{db}^\delta loc_2$ indicates that loc_2 replicates $loc_1^1(db)$ until time δ , so, if permitted, it can serve requests for data in $loc_1^1(db)$. As a consequence, $loc_1 |_{db}^\delta loc_2$ can be rewritten (in *path* operations concerning db) as loc_1 or as $loc_2^2(loc_1)$. The following example shows the use of *Choice*.

EXAMPLE 2. Consider the query of the previous example, and assume that $loc_{11}(db1)$ is replicated at loc_{17} till time δ ; furthermore, assume that the query was submitted at time

τ' so that $\tau' < \delta$. Then, the query can be expressed by the following algebraic expression:

$$return_{entry[\nu \$d, \nu \$p]}(path_{(//, \$b, in)building[(/, \$d, in)desc[0]]}((/, \$p, in)price[0]))(loc_1 \bullet (loc_{11} |_{db1}^\delta loc_{17}) \bullet loc_{13} \bullet loc_{17}^1(db1))$$

4. OPTIMIZATION PROPERTIES

Four main classes of algebraic rewriting rules can be applied to the XPeer query algebra: *classical* equivalences inherited from relational and OO algebras; *path decomposition* rules, which allows the query optimizer to break complex input filters into simpler ones; *equivalences* for query unnesting; and, finally, *rewriting rules* for location operators. For the sake of brevity, only the rules for location operators will be shown in this paper.

Three main rewriting rules can be applied to location operators: *extrusion* of *LocUnion* operations from *path* operations; *simplification* of *Choice* operators; and *introduction* of *Choice* operations.

PROPOSITION 1. Given a database db disseminated on loc_1 and loc_2 , it holds that:

$$path_f((loc_1 \bullet loc_2)^1(db)) = path_f((loc_1)^1(db)) \cup path_f((loc_2)^1(db))$$

This property states that *LocUnion* operations inside *path* operations can be split and distributed across the query; this, in turn, allows the system to more easily decompose a query in single-location subqueries.

PROPOSITION 2. Given a database db hosted at loc_1 and replicated at loc_2 , it holds that:

$$path_f((loc_1 |_{db}^\delta loc_2)^1(db)) = path_f(loc_1^1(db))$$

$$path_f((loc_1 |_{db}^\delta loc_2)^1(db)) = path_f(loc_2^2(loc_1)(db))$$

This property shows how a *Choice* operation inside a *path* operation can be rewritten; we expect that this rewriting will be guided by additional information about network conditions, peer computing power, etc.

PROPOSITION 3. Given a database db , if $(loc_1, loc_2, db, \delta_1, \delta_2) \in replicas$, and $\delta_2 \geq \tau' - \delta_{\tau'}$, then $loc_1^1(db) \rightarrow (loc_1 |_{db}^\delta loc_2)^1(db)$

COROLLARY 1. Given a database db , if $(loc_i, loc_j, db, \delta_1, \delta_2) \in replicas$, and $\delta_2 \geq \tau' - \delta_{\tau'}$, then

$$path_f((loc_1 \bullet loc_i)^1(db)) \rightarrow path_f((loc_1 \bullet (loc_i |_{db}^\delta loc_j))^1(db))$$

These properties back the introduction of *Choice* operations in query plans.

5. CONCLUSIONS

This paper presented a query algebra for XML p2p databases. The query algebra features mechanisms for modeling data dissemination and replication, and for incorporating replication constraints into query plans. The proposed algebra can be used for expressing query plans, as well as for formally reasoning about query semantics in a p2p environment, and, in particular, for investigating correctness and completeness properties of query results.

6. REFERENCES

- [1] Sophie Cluet and Guido Moerkotte. Classification and optimization of nested queries in object bases. Technical report, University of Karlsruhe, 1994.
- [2] Denise Draper, Peter Fankhauser, Mary Fernandez, Ashok Malhotra, Kristoffer Rose, Michael Rys, Jérôme Siméon, and Philip Wadler. XQuery 1.0 and XPath 2.0 Formal Semantics. Technical report, World Wide Web Consortium, August 2003. W3C Working Draft.