ROLEX: Relational On-Line Exchange with XML

Philip Bohannon
Bell Labs, Lucent Technologies
bohannon@lucent.com

Xin (Luna) Dong
University of Washington
lunadong@cs.washington.edu

Sumit Ganguly
Bell Labs, Lucent Technologies
sganguly@lucent.com

Henry F. Korth
Lehigh University
korth@cse.lehigh.edu

Chengkai Li
University of Illinois at Urbana-Champaign
cli@uiuc.edu

P.P.S. Narayan
Bell Labs, Lucent Technologies
ppsnarayan@lucent.com

Pradeep Shenoy
University of Washington
pshenoy@cs.washington.edu

1. OVERVIEW

ROLEX[1] is a research system for closely coupled XML-relational interoperation [2]. Whereas typical XML-based applications interoperate with existing relational databases via a "shred-and-publish" approach, the ROLEX system seeks to provide direct access to relational data via XML interfaces *at the speed of cached* XML *data*. To achieve this, ROLEX is integrated tightly with both the DBMS and the application through a standard interface supported by most XML parsers, the Document Object Model (DOM). Thus, in general, an application need not be modified to be used with ROLEX. With the DBMS providing performance qualitatively similar to cached data, XML applications can rely on it for concurrency control and recovery services. To support our integration model and performance goals, ROLEX is built on the DataBlitz^{TM} Main-Memory Database System, allowing us to capitalize on low-latency access to data while still providing concurrency control and recovery [1].

The DOM interface supports the expected navigation functions: parent-to-child, child-to-parent, and sibling-to-sibling. A DOM interface to an XML view query supports all the DOM operations and behaves as if the user were navigating the XML document resulting from the query. In ROLEX, we implement a *virtual* DOM tree that goes a step further by providing the same interface without creating a physical DOM tree.

A novelty of ROLEX is that it uses a navigational profile for the user or application when it optimizes view-query plans. While navigational profiles can, in principle, be quite complex, we currently adopt a very simple model. If n is a node in the schema tree with parent p, the navigation profile stores the probability that some node in the DOM tree generated by n will be visited given that its parent, generated by p, has been visited.

Query output is produced by a *navigable query plan*. It provides, for each node n in the schema tree of a view query, two entities: (1) a *subplan* for evaluating the tag query for n, and (2) a *navigation index*. The navigation index serves to materialize the output of the tag query and supports efficient lookup based on parameter values, as well as navigation to siblings to support tree traversal. The subplan may populate the navigation index lazily or eagerly as decided by the optimizer (based on the navigational profile), and it may also materialize results to be used by other subplans.

The execution engine has been built to serve as a general in-memory relational query-execution engine, as well as the execution engine for ROLEX. The engine handles a variety of join techniques, group-by and aggregates, and the materialization options discussed in [2]. In the implementation, the engine is decoupled from the optimizer, and an XML plan representation is used to communicate between the two.

[1]ROLEX stands for Relational On-Line Exchange with XML. The authors' work presented here was done while they were with Bell Labs.

2. DEMONSTRATION

The demonstration has a browser-based interface through which queries can be entered and edited. Once the user is satisfied, a query plan is generated along with a graphical (postscript) representation of the plan. Query results are displayed using a standard browser. While the gathering of navigational statistics is not implemented in the demonstration, the user may enter the profile probabilities manually with the query to see the impact on plan generation.

We have implemented simple update functionality to demonstrate the novel potential of updating relational data directly using the DOM interface. This implementation is part of ongoing work in defining updates through XML views [3].

We demonstrate XSLT stylesheets running on the DOM result of view queries. However, this demonstration does not currently allow all features of XPATH and XSLT. We expect also to demonstrate a preliminary version of an XSLT view-composition algorithm that is the subject of on-going research [4]. The idea of this algorithm is to push as much functionality as possible from the XSLT stylesheet to the underlying query engine, which is expected to yield substantial performance benefits.

3. REFERENCES

[1] J. Baulier et al. DataBlitz storage manager: Main memory database performance for critical applications. In *Proc. of the ACM SIGMOD Int'l. Conf. on the Mgmt. of Data*, 1999. Industrial track paper.

[2] P. Bohannon, S. Ganguly, H. F. Korth, P. P. S. Narayan, and P. Shenoy. Optimizing view queries in ROLEX to support navigable result trees. In *Proc. 28th Int. Conf. Very Large Data Bases, VLDB*, pages 119–130, 2002.

[3] X. L. Dong, P. Bohannon, H. F. Korth, and P. Narayan. Updating XML views of relational data. In *(submitted for publication)*, 2003.

[4] C. Li, P. Bohannon, H. F. Korth, and P. Narayan. Composing XSLT transformations with XML publishing views. In *Proc. of the ACM SIGMOD Int'l. Conf. on the Mgmt. of Data*, 2003.