An XML-based Framework for Temporal Database Implementation*

Seo-Young Noh and Shashi K. Gadia
Department of Computer Science
Iowa State University
Ames, IA, 50010, USA
{rsyoung, gadia}@cs.iastate.edu

Abstract

This paper presents an XML-based approach to implementing the parametric model of temporal databases. In the parametric model, attribute values are functions of time and the entire history of an object is modeled in terms of a single tuple. This property makes it difficult to adapt the parametric model within the conventional databases. However, XML is an attractive option for implementation because data boundaries are not problematic in XML. Native XML storage with pagination is used for storing temporal data and the DOM parser for the paginated XML storage is used for data access. The primitives for the query language are implemented using the DOM parser. Many artifacts, such as parse tree and expression tree are also represented in XML. We hope that this paradigm offers an elegant approach for implementation of complex data models.

1. Introduction

The parametric data model is the base model in our implementation. It uses *temporal element-based*, *explicit time domain*, and *attribute-level timestamping* schemes. In the parametric model, attribute values are functions of time and the entire history of an object is modeled in terms of a single tuple. This feature is one of the main differences between the parametric data model and most other temporal data models that capture object characteristics in terms of multiple tuples. Tansel [4] is a notable exception that also models an object in a single tuple. Due to this property, the parametric data model reduces query complexities without introducing self-join operations for combining tuples on an object. In spite of the advantages, it increases the implementation complexity because attribute sizes are not predefined.

*This work has been sponsored in part by Information Infrastructure Institute and a grant from Baker Foundation Trust at Iowa State University.

Over the past few years, many researchers have utilized XML technology in databases. Because of XML, a new database generation is emerging. Temporal databases are no exception. Because of XML's flexible modeling mechanism, it is possible to describe complex characteristics of temporal data.

In this paper, we introduce an XML-based implementation of temporal database system for the parametric data model. All mechanisms used in the database system are based on XML technology. However it differs from general XML database systems in that it has its own temporal query language called ParaSQL.

2. Parametric Data Model for Temporal Data

Figure 1 shows an example of a temporal parametric relation. The relation maintains the history of departments.

DName		MName	
[11,49]	Hardware	[11,44]	John
		[45,49]	Leu
[41,47]	Software	[41,47]	Tom
∪ [71,NOW]		[71,NOW]	Inga

Figure 1. Dept relation

Temporal Elements: To obtain timestamps that are closed under the set theoretic operations of *union*, *intersection* and *complement*, the concept of *temporal elements* is introduced in the parametric data model [1]. In Dept relation, an example of temporal elements is [41,47] ∪ [71,NOW].

Temporal Attribute Values: To capture the changing value of an attribute, a *temporal value* of an attribute is defined as a function from a temporal element into the domain of the attribute [1]. The temporal value of MName attribute in the first tuple is ⟨[41,47] Tom, [71,NOW] Inga⟩. If ξ is

Proceedings of the 12th International Symposium on Temporal Representation and Reasoning (TIME'05)
1530-1311/05 $20.00 © 2005 **IEEE**

a temporal value, $[\![\xi]\!]$ denotes its domain. Thus $[\![\langle$[41,47] Tom, [71,NOW] Inga$\rangle]\!]$ =[41,71] ∪ [71,NOW].

XML Representation: The following shows an XML representation of Dept relation.

```
<Rel name="Dept">
  <tup>
    <attr name="DName">
      <val>Hardware<dom><int>[11,49]</int>
                    </dom></val></attr>
    <attr name="MName">
      <val>John<dom><int>[11,44]</int>
                </dom></val>
      <val>Inga<dom><int>[45,49]</int>
                </dom></val></attr>
    <dom><int>[11,44]</int>
         <int>[45,49]</int></dom>
  </tup> <!-- the other is omitted -->
</Rel>
```

Each tuple is mapped to <tup> element. A domain may contain multiple intervals representing a temporal element. It is worth noting that each <tup>, <attr>, and <val> element has its own <dom> element. It is designed to provide rapid responses to domain specific queries. This approach can retrieve domains of objects without stepping through a large part or entire data.

Parametric Structured Query Language: The ParaSQL's select statement is SQL-style select statement. The following shows the BNF of the select statement in ParaSQL.

```
SELECT <attribute list>
[RESTRICTED TO <domain expression>]
FROM <relation list>
[WHERE <boolean expression>]
```

A domain expression is used to restrict the domain of tuples (or objects) filtered by a boolean expression. A boolean expression has the same functionality as classical SQL in that it either qualifies or disqualifies a tuple. But it differs from classical SQL in that it can be constructed by domain expressions with set operations.

3. System Architecture

We have implemented an XML-based temporal database system. The system consists of three layers: Query Processing Layer, Query Execution Layer, and Storage Management Layer. Figure 2 shows the system architecture of our framework.

Query Processing Layer: This layer generates a physical query plan from ParaSQL queries. A ParaSQL query is parsed and the parse tree is internally represented by XML. By using a DOM parser, the parse tree is transformed into an expression tree, which is also represented by XML.

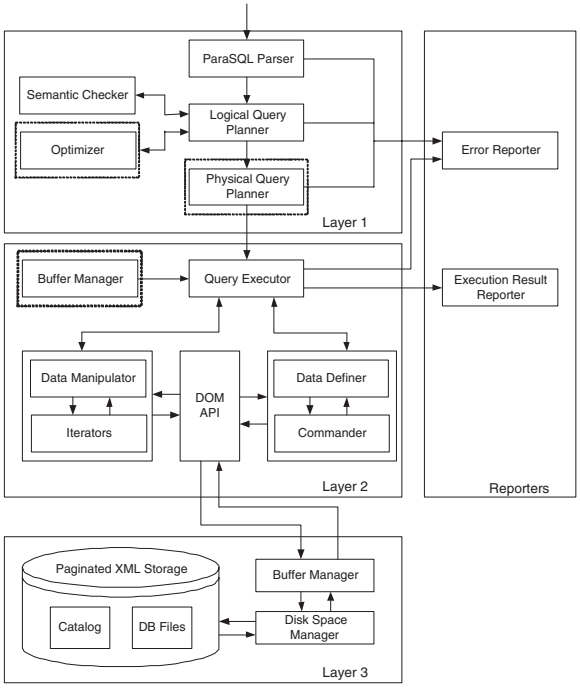

Figure 2. System architecture

Query Execution Layer: This layer executes an expression tree represented in XML. The evaluation functions for RESTRICTED TO and WHERE clauses are the most important functions in query executions. Two clause are handled by EVALUATION and RESTRICTION functions, respectively. These two functions share the same base primitives that call NEXTEVAL function internally. But they are different in that EVALUATION returns either *true* or *false* while RESTRICTION returns a domain. Expression trees contain annotation nodes to provide query execution information. For example, an annotation node may indicate which iterator should be used in the execution.

Storage Management Layer: This layer manages paginated XML data. Whenever it receives a request, it retrieves one page at a time from the disk. DOM API retrieves nodes from loaded pages. Natix [2] is a storage technology for XML documents. We have developed our own storage technology for XML [3]. In order to facilitate pagination of an XML document, Natix as well as our storage technology add some auxiliary nodes to the document. Whereas in Natix a page consists of several (small) XML elements, our pages are self-contained XML documents on their own right. To a client of our DOM API, auxiliary nodes and page boundaries are transparent. The fact that each page is a self-contained XML document has some interesting advantages for system development. A detailed discussion of the storage system is beyond the scope of this paper.

Proceedings of the 12th International Symposium on Temporal Representation and Reasoning (TIME'05)
1530-1311/05 $20.00 © 2005 **IEEE**

4. Experimental Results

Test Data: We created a synthetic XML data of $372,385$ employee tuples, where a single tuple is $2,800$ bytes. The test XML data is paginated into 4KB pages and approximately 70% of each page is occupied by data.

Test Queries: We tested our proposed system with five queries, which are introduced by Wang and Zaniolo [5]. The five queries are as follows: 1) Retrieve entire employee information, 2) Retrieve the salary history of employee 'Bob', 3) Find all employees throughout '1995-05-01' to '1996-04-30', 4) Find employees on '1996-01-31', and 5) Find the history of employees with department information.

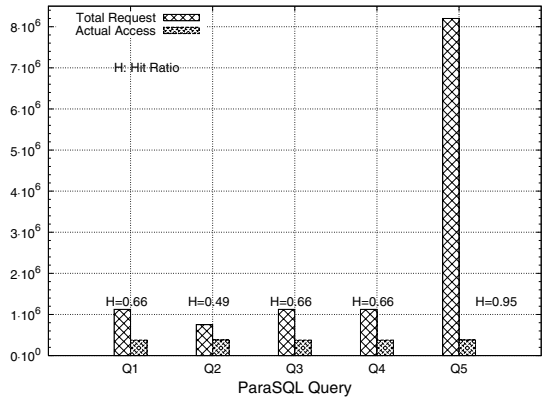

Figure 3. Disk access for 1 GB with 80 buffers

Test Results: Figure 3 shows the performance results for five queries[1]. It is worth noting that the performances for query 1, 3, and 4 are identical. These queries are different types of queries, that is, *relation scan*, *interval*, and *snapshot*. However, they are the same query type in the ParaSQL's viewpoint. For capturing a missing `RESTRICTED TO` clause, ParaSQL sets the domain restriction to *universal time*. The universal time, an interval, and a time instant are all temporal elements in ParaSQL. Therefore three queries have similar expression trees, hence they have the similar performance results.

Even though the 5th query involvs in a join, the number of disk access is comparable to the other queries because the Dept relation is small in our benchmark.

5. Conclusions

We have presented an approach for database implementation using XML. Sometimes the need for designing user-friendly query languages or other reasons leads to pre-fixed data structures that are difficult to implement. XML can elevate such difficulties because of no data boundary limitation. It has also the advantage of eliminating our reliance on linked-list based data structures for parse tree and expression tree analysis.

For the system validation, we tested five different types of queries. The queries were executed on a 1GB database. We noted that although some of these queries seem different, they are quite similar in the parametric data model because the universal time, a time interval, and a time instant are all temporal elements in the parametric data model. Our benchmark for the 5th query can be improved to reflect the nature of join operations more clearly. Benchmarks could also include small to very large tuple spanning several page. However, emphasis in this paper is in showing the usefulness of XML in implementing complex data models.

Despite extensive work in temporal databases, it is difficult to find native implementations that are independent of conventional databases. XML is a growing trend and adapted in many systems and research areas. Temporal databases are no exception. In this paper, we showed XML's usability for native implementation of a temporal database system to fill this void. We hope that this paradigm provides an elegant solution for implementation of complex data models.

Acknowledgments: Authors are thankful to Shihe Ma for his help in preparation of this work.

References

[1] S. K. Gadia. A homogeneous relational model and query languages for temporal databases. *ACM Transactions on Database Systems*, 14, 1988.

[2] C.-C. Kanne and G. Moerkotte. Efficient storage of xml data. In *Proceedings of the 16th International Conference on Data Engineering*, San Diego, CA, USA, Feb. 2000.

[3] S. Ma. Implementation of a canonical native storage for xml. Master's thesis, Department of Computer Science, Iowa State University, Dec. 2004.

[4] A. U. Tansel. A generalized relational framework for modeling temporal data. In *Temporal Databases: Theory, Design, and Implementation*, pages 183–201. Benjamin/Cummings, 1993.

[5] F. Wang and C. Zaniolo. Publishing and querying the histories of archived relational databases in xml. In *Proceedings of the 4th International Conference on Web Information Systems Engineering*, Roma, Italy, Dec. 2003.

[1] We conducted the test with LRU (Least Recently Used) policy.