Temporal Queries in XML Document Archives and Web Warehouses

Fusheng Wang and **Carlo Zaniolo**

Computer Science Department
University of California, Los Angeles
Los Angeles, CA 90095, USA
{wangfsh, zaniolo}@cs.ucla.edu

Abstract

By storing the successive versions of a document in an incremental fashion, XML repositories and data warehouses achieve (i) the efficient preservation of critical information, and (ii) the ability of supporting historical queries on the evolution of documents and their contents. In this paper, we present efficient techniques for managing multi-version document histories and supporting powerful temporal queries on such documents. Our approach consists in (i) concisely representing the successive versions of a document as an XML document that implements a temporally grouped data model, and (ii) using XML query languages, such as XQuery, to express complex queries on the content of a particular version, and on the temporal evolution of the document elements and their contents. We show that the data definition & manipulation framework of XML & XQuery can support temporal models and historical queries significantly better than the traditional framework of Relational Tables & SQL. To demonstrate this point, we investigate how to express complex queries on the history of relational tables published as XML data.

1 Introduction

Version management and temporal queries will find important applications in the next generation of web information systems. A first application domain is XML document archives and digital libraries. Indeed the computing technology that makes digital repositories possible also makes it very easy to revise the documents and publish their revised versions on the Web. To avoid loss of critical information and achieve e-permanence, old versions must be preserved. In the future, we can expect that 'e-permanence' standards will be established for critical Web sites of public interest [5].

A first approach to preserve the content of successive document versions, is storing each version as a separate document. But this is very undesirable because of (i) storage inefficiency, (ii) explosion of hits on content-based searches, and (iii) difficulty of answering queries on the evolution of a document and its content. Therefore, in this paper, we propose XML-based techniques whereby a multi-version document is managed as a unit, and successive versions are represented by their delta changes to optimize storage utilization, and support efficiently complex historical queries on the evolution of the document and its elements (e.g., *abstract*, *figures*, *bibliography*, etc.).

Similar problems and opportunities occur in warehousing applications, where the need for Temporal Data Warehouses is well-established [39, 30]. Furthermore, as we move from traditional inter-company warehouses to intra-company warehouses, reliance on XML and Web-based environments is bound to grow. A related new trend is Web warehouses designed to collect contents from Web sites of interest, and monitor them at regular intervals to detect changes and provide subscribed services to community of users [28]. Typical services provided include (i) detecting changes from the previous version, (ii) forwarding significant deltas to subscribers, (iii) answering continuous queries over the stream of changes, (iv) retrieving old versions, and (v) supporting historical and trend analysis queries. Thus, we focus on information systems, such as digital libraries and Web warehouses, that support sophisticated change management and temporal queries for focused application domains. This differentiates our web warehouses and digital archives from systems, such as the Wayback machine, which are primarily interested in preserving old snapshots of the global Web [25], rather than on providing sophisticated information services.

In addition to these new applications of version management, we find the more traditional ones, such as software configuration and cooperative work [1, 36]. As these applications migrate to a Web-based environment, they increasingly use XML for representing and exchanging information, often seeking standard vendor-supported tools for processing and exchanging their XML documents.

The importance of version management has been fully recognized by XML standard groups [1], but because of the difficult research issues that remain open, standards have not been issued yet. This current situation provides a window of opportunity to solve the technical challenges and lay the foundations for this important piece of Web information systems technology.

The first problem that must be addressed is how to represent a multiversion document as an XML document that (i) can be viewed by conventional XML browsers at remote sites, and (ii) can support complex queries, including temporal ones, expressed in standard XML query languages, such as XQuery [2]. For instance, the reference-based representation proposed in [15] achieves the first objective but falls short of the second one. On the other hand, the model based on durable node numbers and document shredding presented in [17] was proven effective at delivering good performance at the physical level [16]; however the problem of how to represent the version history as XML document for querying and/or forwarding it to a remote browser was never solved.

Therefore, the question of what data representation should be used for time-changing content constitutes a difficult problem—also one that greatly influences the design of the temporal query language to be used. For relational databases, more than forty different approaches were counted [40], each featuring a different combination of a temporal data model and query language. Although the design space of alternatives has been so extensively explored, as today, no temporal data model and query language exists which is generally accepted in the research community, or supported by major vendors.

However, as shown in [37], the troublesome experience of relational systems with temporal information is due to the flat structured relational tables, and it does not necessarily carry over to XML, which, e.g., supports a temporally grouped data model that has long been recognized as the most natural and expressive for temporal information [21].

This paper is organized as follows. Section 2 reviews previous work. Section 3 proposes our new scheme to represent XML document changes and support complex queries. Section 4 discusses the advantages to apply our scheme to XML-published relational data. Efficient implementations are discussed in Section 5 and Section 6 concludes the paper.

2 Previous Work

The problem of document history management combines the issues of document version management and temporal databases, for which a large number of models and techniques have been proposed. We will briefly discuss these topics with a focus on the representation of temporal XML documents.

Temporal XML Representation. A *valid* tag for XML/XHTML documents is proposed in [26] to support valid time on the Web, thus temporal visualization can be shown on web browsers. A data model is proposed for temporal XML documents in [6]. In this approach, XML queries can be supported in DOM or XPath after these have been suitably extended. Similarly, TTXPath [22] is extension of XPath data model and query langauge to support transaction time semantics. (We instead support XPath/XQuery without any extension to XML query languages or data models in our approach.)

An interesting archiving technique for scientific data is presented in [9], a scheme based on an extension of the SCCS [31] scheme. In [9], a document is viewed as an unordered set of elements, and an in-depth discussion is presented on how to uniquely identify and retrieve elements by their logical keys; however support for queries (historical or otherwise) is not discussed. Moreover, elements have timestamps only if they are different from the parent nodes, a fact that complicates the task of writing queries in XPath/XQuery. In this paper, we will introduce a scheme that presents several similarities to [9], but provides full support for XML query languages such as XPath and XQuery.

For an XML Warehouse, Web sites of interest are periodically consulted, and the difference between the latest version and the previous one is computed as edit scripts for storing and querying multiversion documents. Some of the efficient change detection algorithms are briefly discussed next.

Change Detection. *LaDiff* [11] is a change detection algorithm for semistructured information, and approaches the problem by dividing it into (i)the *Good Matching* problem and the (ii) *Minimum Conforming Edit Script* problem. A diff algorithm *XyDiff* for XML documents is proposed in [28, 23]. To match the largest identical parts of both documents, the algorithm utilizes ID attribute information, and a signature and a weight is computed for each node from bottom-up. X-Diff [38] detects changes in XML documents based on an unordered model, which applies to published relational data. By utilizing *node signatures* and *node XHash values*, the algorithm tries to find the minimum-cost matching.

Different approaches for supporting multiversion documents use different schemes for storing these deltas on secondary storage and processing them for version reconstruction and query execution. These are discussed next.

Managing the Deltas. The RCS [35] scheme stores the most current version plus reverse editing scripts to retrieve previous versions. An improvement to RCS was proposed in [14], which introduced a temporal clustering scheme to improve the efficiency of retrieving past versions from secondary store. These approaches lack an efficient logical representation to support complex queries [14]. RBVM [15] unifies the logical representation and the physical one, and objects that remain unchanged are stored as references to the objects in old versions. However, this scheme can only handle simple queries; in fact, different storage representations are needed for more complex queries [17, 16]. Another scheme often used in the past for version control is SCCS [31]. In SCCS, each text segment is clustered with all its successive changes; also pairs of timestamps are associated with the segments and their changes to specify their lifespans. Version retrieval is performed by scanning the file and retrieving valid segments according to timestamps. At the physical level, this is a source of much inefficiency, since reconstruction of a single version now requires the complete scan of the file which becomes large and larger as successive versions are added. The addition of indexes [31], only improves the situation to a point, since the segments belonging to the same version are not clustered together; thus retrieving a version can require accessing a different page for each of its segments [14].

Temporal Databases The management of multiple database versions has received much attention under the topic of transaction-time temporal databases [40]. A large number of temporal data models was studied and the design space for the relational data model has been exhaustively explored [29]. Clifford et al. [21] classified them as two main categories: *temporally ungrouped* and *temporally grouped*. Temporal groups are however difficult to support in the framework of flat relations and SQL.

Object-oriented temporal models are compared in [32], and a formal temporal object-oriented data model is proposed in [10]. The problem of version management in object-oriented and CAD databases has receive a significant amount of attention [8, 24]. However, support for temporal queries is not discussed, although query issues relating to time multigranularity were discussed in [12]. Schema versioning represents another important topic frequently discussed in the context of object-oriented databases [40].

3 A Logical Model for Versions

For all the shortcomings of SCCS scheme at the physical level, we will show next that the scheme has some redeeming quality at the logical level. These are not obvious in conventional systems, where the basic document segments managed by SCCS are lines of text. In this situation, the multiversion document generated by SCCS lacks any obvious logical structure. However, when applied to the elements of a well-structured XML document, the basic SCCS scheme can be adapted to produce a well-structured XML document that can be used to (i) display the version history of the document on a Web browser, and (ii) express complex queries on the document and its evolution. Later in the paper, we will show that there is a simple mapping between this representation and the SPaR-based storage scheme proposed in [27]. This provides an answer to most performance related issues.

We now discuss how to summarize and represent the successive versions of a document as an XML document, called a *V-Document* (Figure 2), upon which complex queries and temporal queries can be specified using languages such as XPath or XQuery. In a *V-Document* each element is assigned with two attributes *vstart* and *vend*, which represent the valid version interval (inclusively) of the element. In general, *vstart* and *vend* can be version numbers or timestamps: *vstart* represents the initial version when the element is first added to the XML document, and *vend* represents the last version in which such an element is valid. After the *vend* version, the element is either removed or changed. The value of *vend* can be set to *now*, to denote the ever-increasing current version number. It is clear that the version interval of an ancestor node always contains those of its descendant nodes.

There are significant advantages of our scheme, including the following:

- There is no storage redundancy for multiple version documents, since identical nodes are temporally grouped and represented by valid version intervals;
- The changes are naturally represented with an automatically generated DTD which preserves the hierarchical structure of the original XML document; and
- Temporal queries and other complex queries can be easily expressed on V-documents using standard XML query languages.

3.1 Change Management

We now consider a very simple document and its successive versions as shown in Figure 1.

For simplicity, the following primitive change operations are considered: DELETE, INSERT and UPDATE. (Operations such as MOVE or COPY can be reduced to these). The change of an element is determined by XML-Diff algorithms. The following is the effect of performing each operations:

UPDATE. When an element is considered updated, a new element with the same name will be appended immediately after the original element; the attribute *vstart* of this new element is set to the current version number, and the attribute *vend* is set to the special symbol '*now*' that represents the ever-increasing current version number. The *vend*

```
<document>                      <!--comment: This is version 1 -->
    <chapter no="1">
        <title>Introduction</title>
        <section>Background</section>
        <section>Motiviation</section>
    </chapter>
</document>
<document >                     <!--comment: This is version 2 -->
    <chapter no="1">
        <title>Introduction and Overview</title>
        <section>Background</section>
        <section>History</section>
    </chapter>
    <chapter no="2">
        <title>Related Work</title>
        <section>XML Storage</section>
    </chapter>
</document>
<document>                       <!--comment: This is version 3-->
    <chapter no="1">
        <title>Introduction and Overview</title>
        <section>Background</section>
        <section>History</section>
    </chapter>
    <chapter no="2">
        <title>Related Work</title>
        <section>XML Indexing</section>
    </chapter>
</document>
```

Figure 1. Sample Versioned XML Documents

attribute of the old element is set to the last version before it was changed. The change of an element is not viewed as a change of its ancestors, unless the ancestors themselves are changed. This is consistent with the results produced by the XML-Diff algorithms, where the only deltas listed are those of the changed elements.

INSERT. When a new element is inserted, that element is inserted into the corresponding position in the *V-Document*; the *vstart* attribute is set to the current version number, and *vend* is set to *now*.

DELETE. When an element is removed, the only information that must be changed is the *vend* attribute; this is set to the last version where the element was valid.

3.2 Handling Attributes

In the previous examples, we have assumed that the elements of our document contain no attribute. Elements containing attributes can be easily supported in our V-Document by representing each attribute by a subelement denoted by a special flag attribute `isAttr`. For instance, if the `employee` element contains the attribute `empno` then we represent them as follows:

```
<empno isAttr="yes" vstart="1996-01-01"
vend="now">e1</empno>
```

This transformation, and also the inverse transformation from child element to attribute, is simple and can be implemented in XQuery or XSLT [3].

3.3 DTD of the V-Document

One significant advantage of our scheme is that it keeps the hierarchical structure of original XML documents, and it has a well-defined DTD for the *V-Document*. For example, the original DTD of the employee XML document from the Web is shown in Figure 3. The DTD of *V-Document* can be automatically generated as follows (Figure 4). Each element is added two new attributes *vstart* and *vend*; an attribute of an element will be converted as a child element; and child elements are set as repeatable for different version intervals.

4 Complex Queries

Using our change representation scheme, complex queries can be expressed easily. The following is a list of queries expressed on XQuery [2]. All queries were tested with Quip [4], an XQuery engine implemented by SoftwareAG.

QUERY 1. Snapshot queries: retrieve the version of the document on 2002-01-03:

```
for $e in document("V-Document.xml")
```

```
<document vstart="2002-01-01" vend="now">
    <chapter vstart="2002-01-01" vend="now">
        <no isAttr="yes" vstart="2002-01-01" vend="now">1</no>
        <title vstart="2002-01-01" vend="2002-01-01">Introduction</title>
        <title vstart="2002-01-02" vend="now">Introduction and Overview</title>
        <section vstart="2002-01-01" vend="2002-01-01">Motivation</section>
        <section vstart="2002-01-01" vend="now">Background</section>
        <section vstart="2002-01-02" vend="now">History</section>
    </chapter>
    <chapter vstart="2002-01-02" vend="now">
        <no isAttr="yes" vstart="2002-01-02" vend="now">2</no>
        <title vstart="2002-01-02" vend="now">Related Work</title>
        <section vstart="2002-01-02" vend="2002-01-02">XML Storage</section>
        <section vstart="2002-01-03" vend="now">XML Indexing</section>
    </chapter>
</document>
```

Figure 2. XML Representation of Versioned Document

```
<!ELEMENT document (chapter)*>
<!ELEMENT chapter (title, (section)*)>
<!ATTLIST chapter no CDATA #REQUIRED>
<!ELEMENT title (#PCDATA)>
<!ELEMENT section (#PCDATA)>
```

Figure 3. DTD of original XML documents

```
    /document
return snapshot( $e,"2002-01-03")
```

Here, *snapshot($node, $versionTS)* is a recursive XQuery function that checks the version interval of the element and only returns the element and its descendants where $vstart \leq versionTS \leq vend$.

QUERY 2. Change queries: retrieve the unchanged elements between the version on 2002-01-01 and the version on 2002-01-03:

```
for $e in document("V-Document.xml")
    /document
return diff-identical(
    $e,"2002-01-01","2002-01-03")
```

Here *diff-identical($node, $version1TS, $version2TS)* is another recursive XQuery function that returns the element and its descendants where $vstart \leq version1TS \leq vend$ and $vstart \leq version2TS \leq vend$.

QUERY 3. Evolutionary queries: find the history of the title of chapter 1:

```
for $title in document("V-Document.xml")
    /document/chapter[no="1"]/title
return  $title
```

QUERY 4. Continuous queries: find titles that didn't change for more than 2 consecutive years.

```
for $title in document("V-Document.xml")
    /document/chapter/title
let $duration := substract-dates(
    $title/@vend, $title/@vstart)
where dayTimeDuration-greater-than(
    $duration, "P730D")
return  $title
```

QUERY 5 and 6 in the following show the support of **since** and **until** connectives, a first-order temporal logic extension [20].

QUERY 5. A Since B: find chapters in which section "History" remains unchanged since the version when the title was changed to "Introduction and Overview".

```
for $ch in document("V-Document.xml")
    /document/chapter
let $title := $ch/title[
    title="Introduction and Overview"]
let $sec := $ch/section[.="History"]
where not empty($title) and not empty($sec)
    and $sec/@vstart = $title/@vend
    and $sec/@vend ="now"
return  $ch
```

QUERY 6. A Until B: find chapters in which the title didn't change until a new section "History" was added.

```
for $ch in document("V-Document.xml")
    /document/chapter
let $title := $ch/title[1]
```

```
<!ELEMENT document (chapter)*>
<!ATTLIST document vstart CDATA #REQUIRED vend CDATA #REQUIRED>
<!ELEMENT chapter ( (no)*, (title)*, (section)* )>
<!ATTLIST chapter  vstart CDATA #REQUIRED vend CDATA #REQUIRED>
<!ELEMENT no (#PCDATA)>
<!ATTLIST no isAttr CDATA #IMPLIED vstart CDATA #REQUIRED vend CDATA #REQUIRED>
<!ELEMENT title (#PCDATA)>
<!ATTLIST title vstart CDATA #REQUIRED vend CDATA #REQUIRED>
<!ELEMENT section (#PCDATA)>
<!ATTLIST section vstart CDATA #REQUIRED vend CDATA #REQUIRED>
```

Figure 4. DTD of V-Document

```
let $sec := $ch/section[.="History"]
where not empty($title) and not empty($sec)
    and  $title/@vend = $sec/@vstart
return  $ch
```

JOIN QUERIES. Temporal joins on multiple XML documents can be expressed in standard XQuery, with additional conditions to compare if the nodes to be joined have overlapping version timestamps.

Due to the temporarily grouped features of our model [21], we can express powerful temporal queries, and the need for coalescing is often avoided.

In the above queries, attributes which are represented as children elements can be easily converted back to attributes, and *vstart* and *vend* can also be filtered in the result as needed.

5 XML-Published Relational Data

Relational Database Management Systems (RDBMSs) are now the most popular database systems, and there is huge amount of information stored in relational tables. To support information exchange via XML, relational tables are published as XML documents, which can then be stored in XML document warehouses. Thus temporal queries on the evolution of the database tables can now be expressed in XQuery on these web data warehouses.

Figure 5 shows a sample table of the history of employees, summarizing the changes of the `employee` relation over time. Assuming that the snapshots of this relation were published as XML documents, and the history of these documents were kept in our data warehouse, then we obtain the XML document shown in Figure 6.

The representation used in Figure 6 is a temporally grouped data model [21]. Clifford, et al. [21] show that temporally-grouped models are more natural and powerful than temporarily-ungrouped ones. Temporal groups are however difficult to support in the framework of flat relations and SQL. Thus, many approaches were instead proposed that on timestamping the tuples of relational tables. These approaches incur into several problems, including the coalescing problem [40]. TSQL2's approach [40] tries to reconcile the two approaches[21], and is based on an implicit temporal model that suffers from several problems, discussed in [19].

Our model supports temporal grouping by taking advantage of the richer structure of XML documents, and the expressive power of XQuery. Figure 6 is the XML representation of the temporal relations based on our model. (Although not needed at the logical level, we also show here the attributes *nstart* and *nend*; these are durable node numbers used for the implementation, as discussed later in Section 6. These numbers are only needed for supporting queries on these documents, not for expressing them.)

There are obvious advantages in our approach, which allows us to express temporal queries in XQuery without requiring the introduction of special temporal constructs in the language. For instance, the basic temporal projections, snapshot queries and joins on the *employee* relation(name,title,salary,dept) and the *dept* relation (name, chair) can be expressed as follows:

Q1. Temporal projection: retrieve the salary of employee "Bob" on 1996-01-31:

```
for $s in document("employees.xml")
    /employees/employee[name='Bob']/salary
return snapshot( $s, "1996-01-31")
```

Q2. Snapshot queries: retrieve the employees on 1996-01-31:

```
for $e in document("employees.xml")/employees
return snapshot( $e, "1996-01-31")
```

Q3. Joins: find the chair of employee 'Bob' on 1996-01-31:

```
for $e in document("employees.xml")
    /employees/employee[name='Bob']
for $d in document("depts.xml")/depts/dept
let $et := snapshot($e,"1996-01-31")
let $dt := snapshot($d,"1996-01-31")
where not empty($et) and not empty($dt)
    and  $et/dept = $dt/name
return $dt/chair
```

Name	Salary	Title	DateofBirth	Start	Stop
Bob	60000	Assistant Provost	1945-04-09	1995-01-01	1995-06-01
Bob	70000	Assistant Provost	1945-04-09	1995-06-01	1995-10-01
Bob	70000	Provost	1945-04-09	1995-10-01	1996-02-01
Bob	70000	Professor	1945-04-09	1996-02-01	1997-01-01

Figure 5. A sample employee table

```
<employee vstart="1995-01-01" vend="1996-12-31" nstart="200" nend="2400">
  <name vstart="1995-01-01" vend="1996-12-31" nstart="300" nend="500">Bob</name>
  <salary vstart="1995-01-01" vend="1995-05-31" nstart="700" nend="800">60000</salary>
  <salary vstart="1995-06-01" vend="1996-12-31" nstart="700" nend="800">70000</salary>
  <title vstart="1995-01-01" vend="1995-09-30" nstart="1000" nend="1400">
      Assistant Provost </title>
  <title vstart="1995-10-01" vend="1996-01-31" nstart="1000" nend="1400">Provost </title>
  <title vstart="1996-02-01" vend="1996-12-31" nstart="1000" nend="1400">Professor</title>
  <DateofBirth vstart="1995-01-01" vend="1996-12-31" nstart="1600" nend="2000">
      1945-04-09</DateofBirth>
</employee>
```

Figure 6. V-Document of employee relations

6 Efficient Implementation

In the logical model of our scheme, the V-Document is first clustered by the document structure, and then by the change history. In order to achieve efficient support for document retrieval and querying, a different storage organization is needed, as shown in [16]. In fact, the logical arrangement shown in Figure 2 must be supported in physical organization as shown in Figure 7. First, all elements of the first version are stored at the beginning, and the deltas in the following versions are appended. So the storage is first clustered by *vstart*, and secondly by the order of the elements in the document (thus, the opposite with respect to the logical representation).

To efficiently support version reconstruction and complex queries, we improve this basic organization with Usefulness-based clustering and the SPaR node numbering scheme.

Usefulness-based Clustering. UBCC scheme [13, 16] clusters the objects of a version into a new page if the percentage of valid objects in a page(*Usefulness*) falls below a threshold; thus the worst-case I/O cost for version retrieval is greatly decreased. UBCC can be applied to our scheme as well. Since the records for a given version are clustered, reconstructing the document at a version only requires to retrieve the pages that were useful at that version [17]. The retrieved elements are then ordered according to their DNN numbers (discussed next).

Numbering Scheme. An XML document can be viewed as an ordered tree consisting of tree nodes (elements). A pre-order traversal number can be used to identify the elements of the XML tree [41, 27, 33, 17]. The SPaR(Sparse Preoder and Range) numbering scheme [17] uses durable node numbers (DNN, range) that can sustain frequent updates, which is first proposed in [27]. Thus the interval [*dnn(X), dnn(X)+range(X)*] is associated with element X, and we represent it as [*nstart, nend*] in the *V-Document*(Figure 6).

Support for Complex Queries. The use of DNN also facilitates the maintenance of indexes on multiversion documents. In fact by using DNNs, efficient indexing schemes [17] and query processing algorithms [16, 18] can be used to support complex queries on multiversion documents. For instance, multiversion B-Trees (MVBT) [7] indexing is used to support complex queries. The scheme [16] supports conventional and path expression queries.

7 Conclusions

Version management plays a very important role in XML document archives and Web warehouses, and the problem of efficiently managing versioned documents and supporting complex queries poses interesting challenges for database researchers. One of the issues is how to represent a multiversion document as an XML document that can be viewed by XML browsers and support complex queries including temporal ones, expressed in XQuery, the coming XML query standard.

We have shown that indeed the data definition and manipulation framework of XML and XQuery can support temporal models and historical queries significantly better than the traditional framework of relational tables and SQL.

```
<employee vstart="1995-01-01" vend="1996-12-31" nstart="200" nend="2400">
    <name vstart="1995-01-01" vend="1996-12-31" nstart="300" nend="500">Bob</name>
    <salary vstart="1995-01-01" vend="1995-05-31" nstart="700" nend="800">60000</salary>
    <title vstart="1995-01-01" vend="1995-09-30" nstart="1000" nend="1400">
       Assistant Provost</title>
    <DateofBirth vstart="1995-01-01" vend="1996-12-31" nstart="1600" nend="2000">
       1945-04-09</DateofBirth>
</employee>
<salary vstart="1995-06-01" vend="1996-12-31" nstart="700" nend="800">70000</salary>
<title vstart="1995-10-01" vend="1996-1-31" nstart="1000" nend="1400">Provost</title>
<title vstart="1996-02-01" vend="1996-12-31" nstart="1000" nend="1400">Professor</title>
```

Figure 7. Storage organization

Current implementation work includes using the approach described in [17, 16] which is based on Usefulness-based Clustering and using indexing techniques based on Durable Node Number schemes.

Acknowledgments

The authors would like to thank Shy-Yao Chien and Vassilis Tsotras for many inspiring discussions. This research was supported under NSF Grant NSF-IIS 0070135.

References

[1] WebDAV, *"WWW Distributed Authoring and Versioning"*, www.ietf.org/html.charters/webdav-charter.html.

[2] *"XQuery 1.0: An XML Query Language"*, http://www.w3.org/TR/xquery/.

[3] *"XSL Transformations (XSLT)"*, http://www.w3.org/TR/xslt/.

[4] *"Software AG's XQuery prototype Quip"*, http://www.softwareag.com/tamino.

[5] National Archives of Australias policy statement Archiving Web Resources: A Policy for Keeping Records of Web-based Activity in the Commonwealth Government. http://www.naa.gov.au/recordkeeping.

[6] T. Amagasa, M. Yoshikawa, and S. Uemura, *"A Data Model for Temporal XML Documents"*, DEXA, 2002.

[7] B. Becker, S. Gschwind, T. Ohler, B. Seeger, P. Widmayer, *"On Optimal Multiversion Access Structures"*, Proc. of Symposium on Large Spatial Databases, Vol 692, 1993.

[8] David Beech, Brom Mahbod, *"Generalized Version Control in an Object-Oriented Database"*, ICDE 1988, 14-22

[9] P. Buneman and S. Khanna and K. Ajima and W. Tan, *"Archiving Scientific Data"*, ACM SIGMOD, 2002.

[10] Elisa Bertino, Elena Ferrai and Giovanna Guerrini, *"A Formal Temporal Object-Oriented Data Model"*, in EDBT 1996.

[11] S. Chawathe, A. Rajaraman, H. Garcia-Molina, J. Widom, *"Change Detection in Hierarchically Structured Information"*, in Proc. ACM SIGMOD, 1996.

[12] E. Camossi, E. Bertino, G. Guerrini, M. Mesiti,*"Automatic Evolution of Multigranular Temporal Objects"*, in Proc. of TIME02, the Ninth International Symposium on Temporal Representation and Reasoning, Manchester (UK), July 2002.

[13] S.-Y. Chien, V. J. Tsotras, and C. Zaniolo, *"Version Management of XML Documents"*, in WebDB 2000 Workshop.

[14] S.-Y. Chien, V. J. Tsotras, and C. Zaniolo, *"Copy-Based versus Edit-Base Version Management Schemes for Structured Documents"*, in RIDE 2001.

[15] S.-Y. Chien, V. J. Tsotras, and C. Zaniolo, *"Efficient Management of Multiversion Documents by Object Referencing"*, in VLDB 2001.

[16] S.-Y. Chien, V. J. Tsotras, C. Zaniolo and D. Zhang, *"Efficient Complex Query Support for Multiversion XML Documents"*, in EDBT 2003.

[17] S.-Y. Chien, V. J. Tsotras, C. Zaniolo and D. Zhang, *"Storing and Querying Multiversion XML Documents using Durable Node Numbers"*, in WISE 2001.

[18] S.-Y. Chien, Z. Vagena, D. Zhang, V. J. Tsotras, C. Zaniolo, *"Efficient Structural Joins on Indexed XML Documents"*, in VLDB 2002.

[19] C. X. Chen, C. Zaniolo, *"Universal Temporal Extensions for Database Languages"*, in ICDE 1999.

[20] J. Chomicki, D. Toman, and M.H. Böhlen, *"Querying ATSQL databases with temporal logic"*, TODS, 26(2):145-178, 2001.

[21] J. Clifford, A. Croker, F. Grandi,and A. Tuzhilin, *"On Temporal Grouping"*, in Proc. of the Intl. Workshop on Temporal Databases, 1995.

[22] C.E. Dyreson, *"Observing Transaction-Time Semantics with TTXPath"*, in WISE 2001.

[23] Gregory Cobena, Serge Abiteboul, Amelie Marian, *"Detecting Changes in XML Documents"*, in ICDE 2002.

[24] Hong-Tai Chou, Won Kim, *"A Unifying Framework for Version Control in a CAD Environment"*, in VLDB 1986, 336-344.

[25] B. Kahle, Alexa et al., *"The Internet Archive–The Wayback Machine–Surf the Web as it was"*, http://www.archive.org/index.html

[26] F. Grandi and F. Mandreoli, *"The Valid Web: An XML/XSL Infrastructure for Temporal Management of Web Documents"*, ADVIS, 2000.

[27] Q. Li and B. Moon, *"Indexing and querying XML data for regular path expressions"*, in VLDB 2001.

[28] A. Marian, et al., *"Change-centric management of versions in an XML warehouse"*, in VLDB 2001.

[29] G. Ozsoyoglu and R.T. Snodgrass, *"Temporal and real-time databases: A survey"*. in IEEE Transactions on Knowledge and Data Engineering, 7(4):513–532, 1995.

[30] Dimitris Papadias, Yufei Tao, Panos Kalnis, and Jun Zhang, *"Indexing Spatio-Temporal Data Warehouses"*, in ICDE 2002.

[31] M. J. Rochkind, *"The Source Code Control System"*, IEEE Transactions on Software Engineering, SE-1, 4, Dec. 1975, pp. 364-370.

[32] R. Snodgrass, *"Temporal Object-Oriented Databases: a Critical Comparision"*, in Modern Database Systems: The Object Model, Interoperability and Beyond. Addions-Wesley/ACM Press, 1995.

[33] D. Srivastava, S. Al-Khalifa, H.V. Jagadish, N. Koudas, J.M. Patel, Y.Wu, *"Structural Joins: A Primitive for Efficient XML Query Pattern Matching"*, in ICDE 2002.

[34] F. Tian, D. J. DeWitt, J. Chen, and C. Zhang. *"The Design and Performance Evaluation of Various XML Storage Strategies"*, in SIGMOD Record, March 2002.

[35] W. F. Tichy, *"RCS–A System for Version Control"*, Software–Practice&Experience 15, 7, July 1985, 637-654.

[36] Fabio Vitali, *"Versioning hypermedia"*, in ACM Computing Surveys 31(4es): 24 (1999).

[37] Fusheng Wang and Carlo Zaniolo, *"Preserving and Querying Histories of XML-Published Relational Databases"*, in ECDM 2002.

[38] Y. Wang, D. J. DeWitt, and J. Cai, *"X-Diff: A Fast Change Detection Algorithm for XML Documents"*, in ICDE 2003.

[39] Jun Yang, *"Temporal Data Warehousing"*. Ph.D. Dissertation, Stanford University, 2001.

[40] C. Zaniolo, S. Ceri, C. Faloutsos, R. T. Snodgrass, V.S. Subrahmanian and R. Zicari, *"Advanced Database Systems"*, Morgan Kaufmann Publishers, 1997, pp97-160.

[41] C. Zhang, J. F. Naughton, D. J. DeWitt, Q. Luo and G. M. Lohman, *"On Supporting Containment Queries in Relational Database Management Systems"*, Proc. SIGMOD, 2001.