Available online at www.sciencedirect.com
ScienceDirect
Data & Knowledge Engineering xxx (2007) xxx–xxx
DATA & KNOWLEDGE ENGINEERING
www.elsevier.com/locate/datak

Temporal queries and version management in XML-based document archives

Fusheng Wang [a,*], Carlo Zaniolo [b]

[a] Integrated Data Systems Department, Siemens Corporate Research, 755 College Road East, Princeton, NJ 08540, USA
[b] Computer Science Department, University of California, Los Angeles, 405 Hilgard Avenue, Los Angeles, CA 90095, USA

Received 21 September 2006; received in revised form 23 August 2007; accepted 29 August 2007

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 to support 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 of: (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 contents. We show that the data definition and manipulation framework of XML and XQuery can effectively support temporal models and historical queries without requiring extensions to the current standards; in fact, this approach is effective at representing and querying the histories of relational database tables, which are difficult to manage using SQL. These conclusions emerge through a number of interesting case studies presented in this paper that include W3C documents, the UCLA course catalog, and the *CIA World Factbook*.
© 2007 Published by Elsevier B.V.

Keywords: Version management; Temporal queries; XML documents; Web warehouses

1. Introduction

Temporal queries and version management for Web archives are important components of Web information systems. They are particularly important in applications such as document archives and digital libraries that must ensure the permanence of e-documents. Indeed, the very computing technology that makes digital repositories possible also makes it very easy to revise documents and publish the latest versions on the Web. To avoid loss of critical information and achieve e-permanence, old versions must be preserved. We can expect that in the future, "e-permanence" standards will be established for critical Web sites of public interest [17].

* Corresponding author. Tel.: +1 609 203 5207; fax: +1 609 734 6565.
E-mail addresses: fusheng.wang@siemens.com (F. Wang), zaniolo@cs.ucla.edu (C. Zaniolo).

0169-023X/$ - see front matter © 2007 Published by Elsevier B.V.
doi:10.1016/j.datak.2007.08.002

Please cite this article in press as: F. Wang, C. Zaniolo, Temporal queries and version management ..., Data Knowl. Eng. (2007), doi:10.1016/j.datak.2007.08.002

UNCORRECTED PROOF

A first approach to preserve the content of successive document versions is storing each version as a separate document. However, 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 data warehousing applications, where the need for temporal data warehouses is well established [60,49]. Furthermore, as we move from traditional intra-company warehouses to inter-company warehouses, reliance on XML and Web-based environments is bound to grow. A related new trend is Web warehouses (Fig. 1) designed to collect contents from Web sites of interest, and monitor them at regular intervals to detect changes and provide subscribed services to a community of users [44]. Typical services provided include: (i) detecting changes from the previous version, (ii) forwarding significant deltas to subscribers, (iii) answering continuous queries about changes, (iv) retrieving old versions, and (v) supporting historical queries 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 that are primarily interested in preserving old snapshots of the global Web [4], rather than on providing sophisticated information services. We instead want to provide the technology whereby powerful historical queries can be supported along with sophisticated version-auditing techniques such as those supported by the Versioning Machine project [11].

A key problem that will be addressed in this paper is how to represent a multi-version document as an XML document that (i) can be viewed by conventional XML browsers at remote sites and (ii) can also support complex queries, including temporal ones, expressed in standard XML query languages, such as XQuery [15]. Previous works have focused on either problem rather than both: for instance, the reference-based representation proposed in [27] achieves the first objective and it is also amenable to efficient implementation. However, the approach proposed in [27] is not suitable for expressing temporal queries of any sophistication. In general, the question of which data representation should be used for modelling time-dependent information and facilitating temporal queries has generated many unsolved research issues for database researchers. The difficulty of such temporal issues is vividly demonstrated by the number and diversity of approaches that were proposed over the years to solve the problem. In particular, for relational databases, more than 40 different approaches were counted [61]; each featured a different combination of temporal data model and query language. Although the design space of alternatives has been so extensively explored, as of today, no temporal data model and query

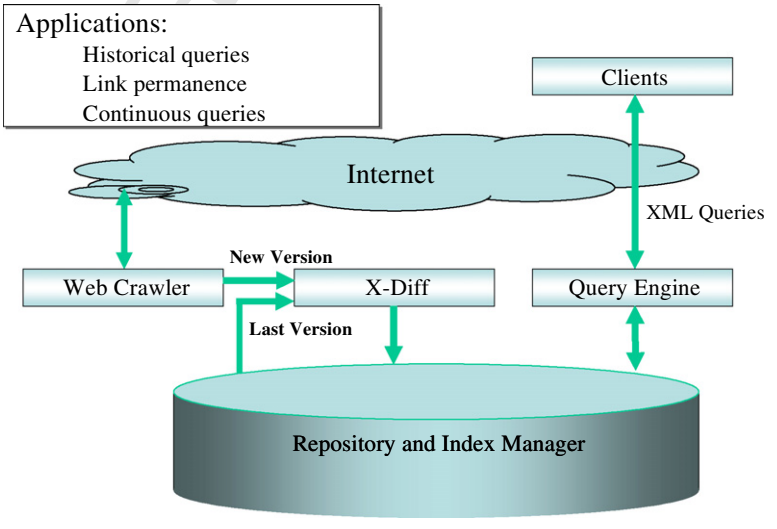

Fig. 1. Web information warehousing.

Please cite this article in press as: F. Wang, C. Zaniolo, Temporal queries and version management ..., Data Knowl. Eng. (2007), doi:10.1016/j.datak.2007.08.002

language exists that is generally accepted in the research community or supported by major vendors. Fortunately, the troublesome experience of relational systems with temporal information is in many ways due to the inflexible flat structure of relational tables, and does not necessarily carry over to XML [55,56]. Indeed XML provides a richer data model, whose structured hierarchy can be used to support temporally-grouped data models that have long been recognized as the most natural and expressive for temporal information [34,33]. Also XML provides a powerful query languages, XQuery, that, unlike SQL, achieves native extensibility and Turing completeness [42]. In fact, we will show in this paper that complex temporal queries on the history of a relational database can be expressed more easily once this history is published (or viewed) in XML.

The use of standard XML in archiving also dovetails with the requirements imposed by more traditional applications of version management, such as software configuration and cooperative work [12,54]. 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 [12], 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.

This paper is organized as follows. Section 2 reviews previous work. Section 3 proposes our new scheme to represent XML document changes, and Section 4 shows how complex temporal queries can be supported on this scheme. Section 5 discusses the general approach of ICAP [10] and two use cases. Section 6 discusses implementation techniques, and Section 7 concludes the paper.

2. Previous work

The problem of document history management combines issues from document version management and temporal databases, for which a large number of models and techniques have been proposed, but often for situations different from the one considered here. For instance, while the problem of preserving the history of an XML document is akin to that of transaction-time relational databases, much of the previous work on temporal queries has been developed in the framework of valid-time databases. Therefore, from this large body of work, we will only discuss previous research that is most relevant to our specific problem, with a focus on the representation of temporal XML documents.

2.1. Temporal XML representations

The problem of supporting valid time on the Web was studied in [41] by Grandi and Mandreoli who proposed a new <valid> markup tag for XML/HTML documents. Rather than temporal queries, the focus of their work was visualization for temporal information, which they showed can be achieved via browsers and XSLT [16]. Likewise, the problem of supporting historical queries was not addressed in [40], where a dimension-based method for managing changes in XML documents was instead proposed. In most previous approaches, the introduction of a new temporal language (or at least of non-trivial time-oriented extensions) is typically regarded as the sine-qua-non for achieving effective support of temporal queries. For instance, Oliboni et al. focus on managing semi-structured temporal data and propose a new language to query documents containing such data [47]. For XML, several extensions have been proposed to the XML data model and its query languages to manage and query temporal information. For instance, extensions to XPath were proposed by Amagasa [18], Dyerson [35], and Mendelzon et al. [46], whereas Gao and Snodgrass proposed the τXQuery language that adds various temporal constructs to XQuery [39]. In our approach, we will instead support temporal representations and queries without any extension to XML and its query languages. An XML-based representations for scientific data was presented in [21]. Our XML scheme presents several similarities with [21], but we also provide full support for historical queries, a topic not discussed in [21].

As discussed in [44,63], maintenance of dynamic warehouses for XML data requires that the Web sites of interest be periodically consulted, and the difference between the latest version and the previous one be computed as edit scripts. The change detection algorithms used to generate such edit scripts efficiently are briefly discussed next.

Please cite this article in press as: F. Wang, C. Zaniolo, Temporal queries and version management ..., Data Knowl. Eng. (2007), doi:10.1016/j.datak.2007.08.002

UNCORRECTED PROOF

2.2. Change detection

LaDiff [23] is a change detection algorithm for semi-structured information that approaches the problem by dividing it into: (i) the *Good Matching* problem and (ii) the *Minimum Conforming Edit Script* problem. The *XyDiff* algorithm for XML documents was proposed in [44,36]. To match the largest identical parts of both documents, the algorithm uses ID attribute information, and a signature and a weight is computed for each node from bottom-up. X-Diff [59] detects changes in XML documents based on an unordered model that is applicable to published relational data. By using *node signatures* and *node XHash values*, the algorithm tries to find the minimum-cost matching. CX-DIFF [38] provides customized change detection of XML content. An interesting diff algorithm that uses relational databases was proposed in [64].

Commercial change detection tools include DeltaXML [3] and Microsoft XMLDiff [5]; these provide tools to generate XML diff and represent it in XML format.

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

2.3. Managing the deltas

The RCS [53] scheme stores the most current version plus reverse-editing scripts to retrieve previous versions. An improvement to RCS was proposed in [26], where a temporal clustering scheme is introduced to improve the efficiency of retrieving past versions from secondary store. These approaches lack an XML-compatible logical representation that can be used to support complex queries [26]. RBVM [27] unifies the logical representation and the physical one by representing objects that remain unchanged 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 [29,28]. Another scheme often used in the past for version control is SCCS [50]. 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 a complete scan of the file which becomes large and larger as successive versions are added. The addition of indexes [50] only improves the situation up 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 [26].

2.4. Temporal databases

The management of multiple database versions has received much attention under the topic of transaction-time temporal databases [61]. A large number of temporal data models were studied, and the design space for the relational data model has been exhaustively explored [48]. Clifford et al. [34] classified them as two main categories: *temporally ungrouped* and *temporally grouped*. The temporally-grouped data model is also referred to as non-first-normal-form model or attribute time stamping, in which the domain of each attribute is extended to include the temporal dimension. Although temporally-grouped models have long been known to be more expressive and appealing to intuition [33,34], they cannot be supported easily in the framework of flat relations and SQL, and therefore they have not been actually implemented in temporal database projects and prototypes [48]. As we will see, temporal data models are instead supported well by XML and its query languages.

Object-oriented temporal models are compared in [51], and a formal temporal object-oriented data model is proposed in [22]. The problem of version management in object-oriented and CAD databases has received a significant amount of attention [20,37]. However, support for temporal queries is not discussed, although query issues relating to time multigranularity were discussed in [24]. Schema versioning represents another important topic frequently discussed in the context of object-oriented databases [61].

The work presented here represents a major extension of the work described in [57]. Thus, this paper presents significant new results, including: (1) extension to arbitrary XML documents by properly handling mixed element and text nodes, (2) detailed explanation of version interval nesting, (3) new discussion on change annotation and visualization, (4) generalized approach to arbitrary documents through significant new case

UNCORRECTED PROOF

Please cite this article in press as: F. Wang, C. Zaniolo, Temporal queries and version management ..., Data Knowl. Eng. (2007), doi:10.1016/j.datak.2007.08.002

studies, including the *CIA World Factbook*, (5) the development of the XChronicler tool to build an XML document that describes document revision history—called V-Document—and the algorithm it uses, (6) examples to demonstrate user-defined functions, (7) detailed storage efficiency analysis of the approach, and (8) XML schemas for sample versioned XML documents.

3. A logical model for versions

Although the SCCS scheme has many shortcomings at the physical level, we will show that it offers potential benefits at the logical level. These are not significant in conventional systems, where the basic document segments managed by SCCS are lines of text, and the multi-version 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 extended to produce a well-structured XML document that can be used to display the version history of the document on a Web browser, and also express complex queries on the document and its evolution.

We now discuss how to summarize and represent the successive versions of a document (Fig. 2) as an XML document, called a *V-Document* (Fig. 3), upon which complex queries and temporal queries can be specified using languages such as XPath or XQuery. In contrast to the relational model, XML provides the opportunity to annotate elements and also have a sequence of such version-annotated elements instead of single separate elements (namely relational tuples). In a V-Document, each element is assigned 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

```
<document>          <!-- Version 1 on 2006-11-01-->
<title>XML-based Scientific Data Modeling</title>
  <section no="1">
    <title>Overview</title>
    <subsection>History</subsection>
    <subsection>The Need</subsection>
  </section>
</document>

<document >         <!-- Version 2 on 2006-11-02 -->
<title>XML-based Scientific Data Modeling</title>
  <section no="1">
    <title>Introduction</title>
    <subsection>History</subsection>
    <subsection>Motivation</subsection>
  </section>
  <section no="2">
    <title>The Data Model</title>
    <subsection>Character of Scientific Data</subsection>
  </section>
</document>

<document>          <!--Version 3 on 2006-11-03 -->
<title>XML-based Scientific Data Modeling</title>
  <section no="1">
    <title>Introduction and Overview</title>
    <subsection>History</subsection>
    <subsection>Motivation</subsection>
  </section>
  <section no="2">
    <title>The Data Model</title>
    <subsection>Metadata Modeling of Scientific Data</subsection>
  </section>
</document>
```

Fig. 2. Sample snapshots of versioned XML documents.

Please cite this article in press as: F. Wang, C. Zaniolo, Temporal queries and version management ..., Data Knowl. Eng. (2007), doi:10.1016/j.datak.2007.08.002

```
<document vstart="2006-11-01" vend="now">
 <title vstart="2006-11-01" vend="now">XML-based Scientific Data Modeling</title>
   <section vstart="2006-11-01" vend="now">
      <no isAttr="yes" vstart="2006-11-01" vend="now">1</no>
      <title vstart="2006-11-01" vend="2006-11-01">Overview</title>
      <title vstart="2006-11-02" vend="now">Introduction</title>
      <subsection vstart="2006-11-01" vend="now">History</subsection>
      <subsection vstart="2006-11-01" vend="2006-11-01">The Need</subsection>
      <subsection vstart="2006-11-02" vend="now">Motivation</section>
   </section>
   <section vstart="2006-11-02" vend="now">
      <no isAttr="yes" vstart="2006-11-02" vend="now">2</no>
      <title vstart="2006-11-02" vend="now">The Data Model</title>
      <subsection vstart="2006-11-02" vend="2006-11-02">Character of Scientific Data</subsection>
      <subsection vstart="2006-11-03" vend="now">Metadata Modeling of Scientific Data</subsection>
   </section>
</document>
```

Fig. 3. V-Document: XML representation of versioned document.

element is either removed or changed. The value of *vend* can be set to *now*, to denote the ever-increasing timestamp value of the current version.

Thus, in our representation, we timestamp every element of our hierarchical structure, where the interval of validity of an ancestor node always contains those of its descendant nodes, since descendant nodes cannot exist without ancestors—e.g., Section 2.1 cannot exist without Section 2. This representation allows parent nodes to exist without children nodes, as is natural in many applications. For instance, as discussed in the later sections, this allows us to represent relational database tables that contain no tuples. For the case of Fig. 3, this means that it will be possible to define sections without giving their titles and the subsections they contain—and thus support useful queries such as "How many sections did the first version of this document contain?".[1]

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

- There is no storage redundancy, since multiple identical nodes which do not change over time in successive versions of the same document are represented as a single node timestamped with its validity interval (an analysis on storage efficiency is discussed in Appendix 2).
- The document history is represented by a temporally-grouped model encoded as an XML document whose DTD (or XML Schema) is automatically generated from the DTD (the schema) of the original document.
- Temporal queries and other complex queries can be easily expressed in 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 Fig. 2.

For simplicity, the only primitive changes used in our V-Documents are DELETE, INSERT and UPDATE. (Operations such as MOVE or COPY can be reduced to these.) These changes are detected by the XML-Diff algorithm and represented in the V-Document by using the *vstart* and *vend* attribute that denote the beginning and end timestamps of the version (or alternatively the version number).

UPDATE. When an element is updated, a new element with the same name will append immediately after the original element; the attribute *vstart* of this new element is set to the current timestamp (or version number), and the attribute *vend* is set to the special symbol *now* that represents the ever-increasing current time-

[1] In certain applications, the implicit constraints of the application might be such that certain nodes never appear without children—e.g., we might not allow sections to be defined unless at least a title is stored for them. In that case, the time intervals of a parent could be computed by coalescing the intervals of its children. But, by timestamping every node in the document hierarchy, we obtain a uniform representation that simplifies querying and maintenance.

Please cite this article in press as: F. Wang, C. Zaniolo, Temporal queries and version management ..., Data Knowl. Eng. (2007), doi:10.1016/j.datak.2007.08.002

UNCORRECTED P

stamp (or version number). The *vend* 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 timestamp (or 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 timestamp (or version number) where the element was valid.

3.2. Annotating and visualizing changes

The XML-based representation proposed here supports well the version control and auditing applications considered in the Version Machine project [11]. In many situations, for instance, it is very desirable to add change annotations that explain the reasons and background for such changes, or to use color coding to visualize changes, possibly between non-successive versions. Now, the V-Document can be easily queried and processed with the XML stylesheet language XSLT [16], to translate into other XML documents or HTML documents, and changes between any two versions can thus be visualized through colors. For instance, three kinds of changes are highlighted: added (yellow), updated (green), and deleted text (strikethrough). The approach is also general and applicable to different types of XML documents. Since the V-Document preserves the structures of the snapshot versions, the XSLT stylesheet for snapshot versions can be reused for version display on the Web.

For example, to mark newly added nodes between Version 1 and Version 2, we can compare each node's version interval, and if the interval is valid at Version 2, and not valid at Version 1, then this is a new node and will be marked with a yellow background with the HTML tag `span`.

However, the most significant benefit achieved from the representation proposed here is that it does a good job of supporting complex queries, particularly historical queries, which are discussed next.

4. Historical queries

Using our change representation scheme, complex queries can be expressed easily. The following is a complete list of temporal queries expressed in XQuery [15].

QUERY 1. Snapshot: show Section 2 of Section 1 as it was on 2006-11-02.

```
for $e in doc("V-Document.xml")/document/section
  [vstart(.) <= "2006-11-02" and "2006-11-02" <= vend(.) and no = "1"]
return ($e/subsection
  [vstart(.) <= "2006-11-02" and "2006-11-02" <= vend(.)]) [2]
```

Here, `vstart($x)` and `vend($x)` are two library functions that, respectively, return the start and end timestamps of the element $x.

QUERY 2. Evolutionary History: show the history of the title of Section 1.

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

QUERY 3. Duration Query: show section numbers and durations for sections that were revised within six months (180 days).

```
for $title in doc("V-Document.xml")/document/section/title
let $ duration :=substract-days(vend($title), vstart($ title))
where duration <= 180
return <section> {$no,$ duration} </section>
```

UNCORRECTED PROOF

Please cite this article in press as: F. Wang, C. Zaniolo, Temporal queries and version management ..., Data Knowl. Eng. (2007), doi:10.1016/j.datak.2007.08.002

The `subtract-days` is a function that computes the differences between dates, recasting the result as days.

In our next query, below, we show how to support the operators **since** and **until** of first-order temporal logic [32].

QUERY 4. A Since B: show the sections which have remained unchanged since 2006-11-01.

```
for $sec in doc("V-Document.xml")/document/section
  and vstart($ sec) <= "2006-11-01" and vend($sec) ="now"
return $sec
```

QUERY 5. A Until B: find the sections in which the title has remained unchanged until a new subsection "Motivation" was added.

```
for $secl in doc("V-Document.xml")/document/section
  for $sec2 in doc("V-Document.xml")/document/section
  let $title= sec2/title[1]
  let $subsec :=$secl/subsection[.="Motivation"]
where not empty($title) and not empty($sec) and
    vend($title) >= vstart($sec)
return $sec2
```

These examples shown above illustrate that the need for coalescing is often avoided or reduced by the use of our temporally-grouped [34] XML representation, on which powerful temporal queries can be conveniently expressed using XQuery.

4.1. Temporal functions

Since XQuery supports user-defined functions, a small library of functions has been defined in our ICAP Project [10] to help users with temporal queries.

For instance, the functions `vstart($e)` and `vend($e)` used in the previous examples were introduced as a convenience to users, but also for data independence since they can hide low-level details used in representing time, e.g., "now." (Internally, we use "end-of-time" values to denote the 'now' symbol. For instance, date 'now' is represented as "9999-12-31.") Other functions that should be defined for user convenience are as follows:

- *Time-interval functions*: `voverlaps($a,$b)`, `vprecedes($a,$b)`, `vcontains($a,$b)`, `vequals($a,$b)`, `vmeets($a,$b)` will return true or false according to two interval positions; `overlapinterval($a,$b)` returns the overlapping interval.
- *Duration and date/time functions*: `vspan($e)` returns the version span of a node, `vstart($e)` returns the start version time of a node, `vend($e)` returns the end version time of a node, and `vinterval($e)` returns the version interval of a node.

More complex user-defined functions were provided to address both the issues of expressive power and user convenience. For instance, we next describe a snapshot function that can be used to construct snapshots of V-Documents of arbitrary structure and nesting:

QUERY 6. Snapshot: retrieve the version of the document on 2006-11-03:

```
for $e in doc("V-Document.xml")/document
return snapshot($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* $\leqslant$ *versionTS* $\leqslant$ *vend*, defined as follows:

UNCORRECTED PROOF

Please cite this article in press as: F. Wang, C. Zaniolo, Temporal queries and version management ..., Data Knowl. Eng. (2007), doi:10.1016/j.datak.2007.08.002

```
define function snapshot($e, $v) {
  if((date($e/@vstart)<=date($v)) and (date($e/@vend)>=date($v)))
  then  element{name($e)} {
    $e/text(),  $e/@*[string(name(.))  !=  "vstart"  and  string(name(.))!=
    "vend"],
    for $child in $e/*
    return snapshot($child, $v)
  }
  else ()
  }
```

Our function defined above can be used to retrieve snapshots of any parts of the document. For instance, to retrieve the snapshot of the titles of Sections 1 and 2 on 2006-12-03, we can use this query:

```
for $d in doc("V-Document.xml")/document
  [vstart(.) <= "2006-12-03" and vend(.) >= "2006-12-03"]
let $sec :=$d/section[no="1" or no="2" and
  vstart(.) <= "2006-12-03" and vend(.) >= "2006-12-03"]
return snapshot($/title, "2006-12-03")
```

Other useful functions in our temporal library include temporal aggregates discussed in the following section.

5. A general approach

The approach proposed in the previous sections is quite general and can be used to preserve (a) the version history of XML documents and (b) the transaction-time history of relational databases. These two application scenarios have been the focus of two UCLA research projects, the ICAP [10] described next and the ArchIS [1] project described in Section 5.4.

5.1. The ICAP Project

Nowadays, many Web documents of practical significance are managed using XML and revised frequently. Thus, we did not have to stray very far to find our first case study: UCLA manages most of its catalogs and other Web documents in XML and updates them periodically. For instance, a new XML version of the UCLA course catalog is published every two years. In this semi-structured document, the courses offered by each department are listed with their numbers, titles, credit units, the grading basis (letter grading or S/U), and the requisites enforced for each course. Along with this structure information, we find a textual description of the topics covered by the course, and other ad hoc annotations. In the ICAP project, we have used the XChronicler tool, to build a V-document that captures the history of recent course catalogs. This V-document makes it possible to use XQuery to pose queries, such as: *Find the new courses introduced by the UCLA CS department in the period 200-2004?* and *How long did it take for keywords such as 'nanotechnology' to migrate from graduate course syllabi to undergraduate syllabi?*.

The use of Web catalogs is ubiquitous in countless services and enterprizes of public and commercial interest and growing every day. Inasmuch as catalogs are frequently revised, the ICAP technology provides a simple but effective approach for enhancing the information systems supporting such catalogs with historical queries and flashback capabilities.

Two other application testbeds studied in the ICAP project were (i) the *CIA World Factbook* that is revised every year [2] and (ii) the history of successive versions for the W3C XLink proposed standards [14].

5.2. Case study: the CIA world factbook

The *CIA World Factbook* is published by the Central Intelligence Agency and has been used as a repository and handbook on population, government, military, and economic information for nations recognized by the

UNCORRECTED PROOF

Please cite this article in press as: F. Wang, C. Zaniolo, Temporal queries and version management ..., Data Knowl. Eng. (2007), doi:10.1016/j.datak.2007.08.002

United States [2]. The report is updated annually and is made public on the Web. However, users can only access the snapshot versions of the *Factbook* from each year, and there is no clue on how the geopolitical and economic situation of the world has evolved over the years. Furthermore, the set of canned queries offered by the *Factbook* Web site is very limited—only keyword search is provided.

5.2.1. Generate structured contents

The *Factbook* documents are currently stored in HTML form; however their regular structure makes it easy to convert them into XML documents. Thus, we built a tool that can effectively crawl the *Factbook* Web pages, extract contents from the pages, and construct a hierarchical XML representation for such pages. Finally, the snapshots from recent years were assembled into a V-Document using the XChronicler described in the next section; the resulting V-Document supports interesting queries on the recent political and economic history of the world. Said V-Document, and the queries discussed next, are available at http://stromboli.cs.ucla.edu/icap/.

5.2.2. Temporal queries on the factbook

The following XQuery examples demonstrate how we can specify structured temporal queries on the V-Document derived from the *World Factbook*.

QUERY FB1. Find the military expenditure history of the United States.

```
for $x in document("factbook04T.xml")/factbook/
  record[country='United States']
return $x/military/military_expenditure_dollar_figure
```

QUERY FB2. Retrieve Canada's GDP on 2002-03-24.

```
for $x in document("factbook04T.xml")/factbook/
  record[country='Canada']
return $x/economy/GDP[vstart(.) <='2002-03-24'
  and vend(.) >='2002-03-24']
```

QUERY FB3. Find Italy's exchange rate history from 2002-02-01 to 2003-04-16.

```
for $x in document("factbook04T.xml")/factbook/
  record[country='Italy']
return $x/economy/exchange_rates[vstart(.) <='2003-04-16'
  and vend(.) >='2000-02-21']
```

5.3. Version History for W3C XLink Specs

The W3C XLink Specs [14] provide an excellent example of the many technical memos and specs that are now published in XML. These text-intensive documents, are frequently revised, because of (i) editorial changes aiming at clarifying and improving the current document and (ii) changes and revisions reflecting the evolution of the technology and systems that these documents are describing.

We have used the XChronicler tool to convert the three versions of XLink specs available to date into a V-document that is managed by the ICAP system. We obtained a simple environment that is supportive of auditing changes and exploring the reasons for such changes. Indeed, changes between successive versions of the documents, or even between non-successive versions of the same, can be easily visualized through colors. Furthermore, annotations can be easily attached to the changed elements in the document to explain the reasons for the changes. Finally, easily-formulated high-level queries can be used to reveal important evolution information such as: *When were certain parts of the specs last revised?*, and *When was the reference to a particular paper first introduced?* We found that the most interesting answers are often returned when histories are que-

UNCORRECTED PROOF

ried for unexpected and improbable events. For instance, when we posed the query, *Find authors who were dropped from the last version of XLink specs*, the only answer returned was "Ben Trafford". We then decided to create an annotation that explains the background for such an unusual change and searched the Web for an explanation of why Ben Trafford was so unceremoniously dropped from the author list of XLink. Initially, we only found reasons for wanting to keep Ben Trafford as a document authors: countless Google entries showed that he was a prolific writer, a popular speaker, and a charismatic technical leader working in the area of electronic books and related Web-technologies. But eventually, after visiting hundreds of links, we found:

A bizarre little tale.... **A Dot-Com Saga** ...

> In US v. Benjamin Trafford, Mr. Trafford was sentenced last Thursday (19 July 2001, after several postponements) to six months in a medium-security federal penitentiary, followed by six months in a halfway house, followed by three years' probation. He will be deported to Canada at the start of probation and barred from re-entering the US. He was also fined $30,000.

Then the Web page http://crism.maden.org/writing/exmu.html proceeds by humorously recounting how, in the heyday of Dot-Com, Ben was drawn into a stream of financial (and salacious) trespasses, culminating to his conviction for stock fraud (thus explaining why he was removed from the XLink author list). This amusing discovery, made possible by the ICAP system [10], illustrates that in real life changes are often more revealing and informative than the status quo. Indeed, the ICAP approach combines the ability of flashing back to previous versions, with that or mining for significant patterns of changes, by using standard XML and its query languages.

5.4. Relational database history: the ArchIS project

The situation of relational databases is also of great practical and scientific interest because of the growing number of time-oriented database applications and because past research on temporal databases has revealed the difficulty of expressing temporal queries on flat relational tables using relations and SQL [61,51,48]. But, rather than restricting ourselves to the relational data model and query languages, we can pursue an approach similar to that we just used for the version history of XML documents. Thus, we can publish the transaction-time history of the relational database in XML and pose temporal queries on such history using standard XQuery. This leads to using a temporally-grouped representation that has long been recognized as the models are more natural and powerful for temporal databases [34]. This representation that could not be easily realized within the rigid structure of flat relational tables can now be supported quite naturally using XML.

While in a relational representation we will have to timestamp individual tuples as shown in Table 1, under a temporally-grouped representation we timestamp columns as shown in Fig. 4, where each attribute value is instead timestamped with its validity period.

The temporally-grouped information of Fig. 4 can be naturally represented using XML as shown in Fig. 5, where each element is timestamped with its validity period by its **tstart** and **tend** attributes. Powerful historical queries can then be naturally expressed against this representation using standard XQuery. No further discussion of these queries will be given here, since they are similar to those we have discussed for V-documents, and several examples have been given in [58]. The XML-published transaction-time history of a relational database can then be stored and queried using a native XML database, as in the case of V-documents.

Alternatively, such histories can be shredded back into relations, and implemented with the help of existing DBMS. This second solution is investigated in [58], where the V-documents were in fact shredded into H-tables, and queries expressed in XQueries which are then mapped into equivalent SQL/XML queries. This makes possible the use of temporal indexing and clustering techniques, that in combination with the support for query optimization of SQL/XML provided by commercial systems, achieves very good levels of performance and scalability [58].

6. Implementation of an XML document versioning system

In this section we discuss how V-Documents can be constructed from the successive versions of arbitrary XML documents and stored for efficient support of temporal queries.

UNCORRECTED PROOF

Please cite this article in press as: F. Wang, C. Zaniolo, Temporal queries and version management ..., Data Knowl. Eng. (2007), doi:10.1016/j.datak.2007.08.002

Table 1
The snapshot history of employees

Id	Name	Salary	Title	Deptno	Start	End
1001	Bob	60,000	Engineer	d01	1995-01-01	1995-05-31
1001	Bob	70,000	Engineer	d01	1995-06-01	1995-09-30
1001	Bob	70,000	Sr Engineer	d02	1995-10-01	1996-01-31
1001	Bob	70,000	TechLeader	d02	1996-02-01	1996-12-31

id	name	salary	title	deptno
1995-05-01	1995-05-01	1995-01-01	1995-01-01	1995-01-01
		60000	**Engineer**	**d01**
		1995-05-31	1995-09-30	1995-09-30
		1995-06-01	1995-10-01	1995-10-01
1001	**Bob**		**Sr Engineer**	
		70000	1996-01-31	**d02**
			1996-02-01	
			Tech Leader	
1996-12-31	1996-12-31	1996-12-31	1996-12-31	1996-12-31

Fig. 4. Temporally-grouped history of employees.

```
<employees tstart="1995-01-01" tend="1996-12-31">
 <employee tstart="1995-01-01" tend="1996-12-31">
  <id tstart="1995-01-01" tend="1996-12-31">1001</id>
  <name tstart="1995-01-01" tend="1996-12-31">Bob</name>
  <salary tstart="1995-01-01" tend="1995-05-31">60000</salary>
  <salary tstart="1995-06-01" tend="1996-12-31">70000</salary>
  <title tstart="1995-01-01" tend="1995-09-30">Engineer</title>
  <title tstart="1995-10-01" tend="1996-01-31">Sr Engineer</title>
  <title tstart="1996-02-01" tend="1996-12-31">Tech Leader</title>
  <deptno tstart="1995-01-01" tend="1995-09-30">d01</deptno>
  <deptno tstart="1995-10-01" tend="1996-12-31">d02</deptno>
 </employee>
 <!-- ... -->
</employees>
```

Fig. 5. The history of the **employee** table is viewed as **employees.xml**.

6.1. XChronicler: generate V-Document from snapshot versions

As shown in Fig. 6, we built a tool named XChronicler [10] to generate the V-Document from the structured diffs taken between successive versions of an XML document (Algorithm 1).

First, the first snapshot version document is normalized as document X_1 by mapping attributes of elements as subelements, and converting mixed text nodes into tagged nodes (discussed in Section 6.2). Document X_1 is then timestamped as document V_1 by adding two attributes *vstart* and *vend*. (Here, V_i represents the composed version history V-Document from version 1 up to version i.)

Then, the second snapshot version document is normalized as document X_2, and the structured $\text{Diff}_{1:2}$ between X_1 and X_2 is computed with MS XMLDiff tool [5]. $\text{Diff}_{1:2}$ is represented as an XML document in Microsoft XML Diff Language, an XML representation of updates. These updates in $\text{Diff}_{1:2}$ are then applied on V_1 to generate the composed history until the timestamp of version 2.

The above process is then repeated until the whole history is composed as V-Document V_n, as per Algorithm 1.

Please cite this article in press as: F. Wang, C. Zaniolo, Temporal queries and version management ..., Data Knowl. Eng. (2007), doi:10.1016/j.datak.2007.08.002

Algorithm 1 XChronicler Algorithm

1: Collect the timestamps of snapshot versions: $T_1, T_2, \ldots, T_n$;
2: Normalize snapshot version 1 as document X_1 by mapping attributes of elements as subelements, and converting mixed text nodes into tagged nodes;
3: Timestamp document X_1 as V-Document V_1 by adding *vstart* and *vend* attributes;
4: i ← 1
5: **repeat**
6: i ← i + 1
7: Normalize snapshot version i as document X_i;
8: Compute the structured diff between document $X_{i-1}$and X_i and generate $\text{Diff}_{i-1:\,i}$;
9: Construct document V_i by applying update actions in $\text{Diff}_{i-1:\,i}$ on V_{i-1};
10: **until**(i > n).

6.2. Handling attributes and mixed content

In our algorithm, we have assumed that the elements of our documents contain no attributes. Elements containing attributes can also be supported in our V-Document by representing each attribute by a subelement denoted by the special tag `isAttr`. For instance, if the `section` elements contain the attribute `no`, then we represent them as follows:

```
<no isAttr="yes" vstart="2002-01-01" vend="now">1</no>
```

This transformation, and also the inverse transformation from child elements to attributes, is simple and can be implemented in XQuery.

Mixed content

We also need to generalize our algorithm to situations where an element can contain text nodes that are mixed with other elements. For example,

```
⟨para⟩⟨b⟩XML⟨/b⟩ is a W3C standard.⟨/para⟩
```

This is handled by transforming a text node to an element `mixtext`:

```
<para><b>XML</b> <mixtext>is a W3C standard.</mixtext></para>
```

With this transformation, all nodes can be associated with version intervals, and similar to attributes, the reverse transformation can be done with XQuery.

With the two extensions discussed above, our approach becomes quite general and applicable to any XML document.

6.3. Schema of the V-Document

One significant advantage of our scheme is that it preserves the hierarchical structure of original XML documents, and it has a well-defined DTD for the *V-Document*.

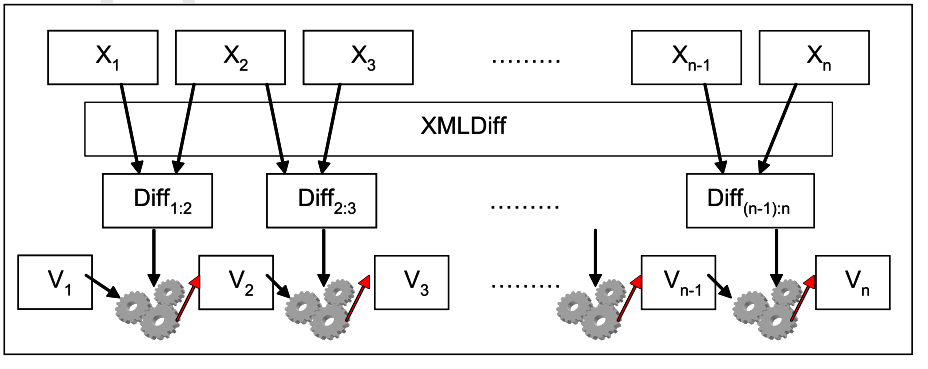

Fig. 6. XChronicler: generate V-Document from snapshot versions.

Please cite this article in press as: F. Wang, C. Zaniolo, Temporal queries and version management ..., Data Knowl. Eng. (2007), doi:10.1016/j.datak.2007.08.002

CORRECTED PROOF

In particular, the DTD for the history of relations can be generated directly for their SQL schema.

As shown in Fig. 7, the DTD of the V-Document (Fig. 8) can be automatically generated from the snapshot DTD. Each element is added with 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.

When a schema is available for the snapshot XML document, the XML schema for the V-Document can be derived in similar fashion, as shown in Appendix 1.

6.4. Efficient storage and retrieval

An important advantage of our approach is that significant savings in storage is achieved because the approach eliminates the duplication of unchanged information that is caused by storing the different snapshots. These savings are realized for both evolving databases and evolving documents. For the first case, we used a database containing the history of employees over 10 years, obtained by introducing regular increases of salaries along with changes of titles and changes of departments. The history is published as a new version represented in XML every 2 years for a consecutive 10 years (five versions). We stored the history in two ways: snapshots and the V-Document. At each version, we computed the total storage up to this version. Fig. 9 shows the comparison of storage between the V-Document and snapshots at different version numbers. As the number of versions increases, the efficiency of the V-Document is apparent. Furthermore, if we increase the number of snapshots, the storage used increases in proportion, while the storage used by the V-Document remains largely unaffected.

As a second example, we tested the version history of W3C XLink standards [14]: there are three versions, published respectively on 2000-07-03, 2000-12-20, and 2000-06-27. The storage is shown in Fig. 10. Observe that the storage savings achieved by the V-Document grow with the number of successive versions.

A practical benefit of our approach is that it can achieve effective support for historical queries by utilizing only current XML and XQuery standards. Thus, any native XML database [8,13] or any XML-extended commercial ORDBMS [6,45] can be used to archive the history of a relational database and a multi-version XML document—as demonstrated by our ICAP and ArchIS systems that are also accessible through the Web [10,1]. (However, for efficiency reasons, the XML database engine should be adapted to support one of the proposed

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

Fig. 7. DTD of snapshot XML documents.

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

Fig. 8. DTD of the V-Document.

Please cite this article in press as: F. Wang, C. Zaniolo, Temporal queries and version management ..., Data Knowl. Eng. (2007), doi:10.1016/j.datak.2007.08.002

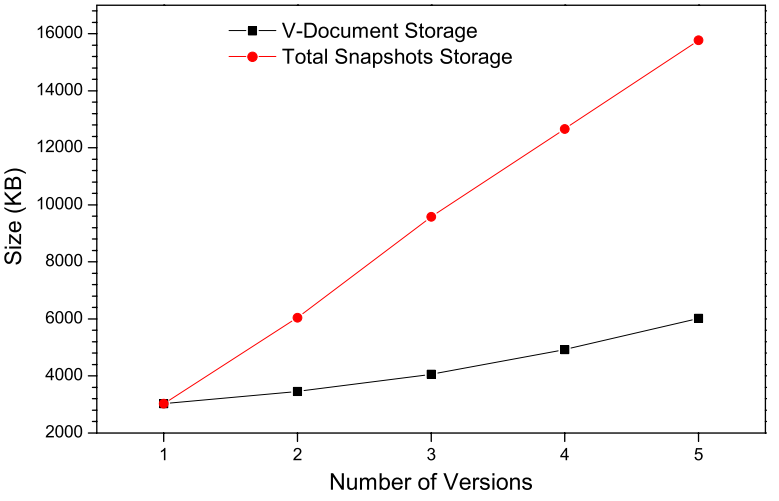

Fig. 9. Employee database: storage used by V-Documents versus snapshots.

clustering/indexing techniques discussed next.) For all its obvious advantages, this solution also inherits the limitations that current native XML databases and XML-extended commercial ORDBMS face—in particular, in terms of scalability since their performance is frequently unsatisfactory when the size of the database grows. Overcoming these limitations represents an important topic of research and the objective of much R&D effort by commercial vendors; However, we will not discuss this general and vast topic since that will take us well beyond the scope of this paper. We will only briefly mention some of the specialized clustering and indexing techniques that have been proposed for improving the performance of temporal queries on multi-versioned documents, and also mention that, for relational table histories, the scalability and performance issues can be solved effectively by shredding the V-Documents back into relational tables.

6.4.1. *Usefulness-based clustering*

This is a powerful technique that can turn a clustering scheme for a single version into a temporal clustering scheme for the multi-version V-Document. For instance, say that we want to minimize the number of disk pages needed to retrieve data concerning all employees working in a given department. Thus, we can start by clustering a single version on `deptno`. However, it is not clear how to store the later deltas describing

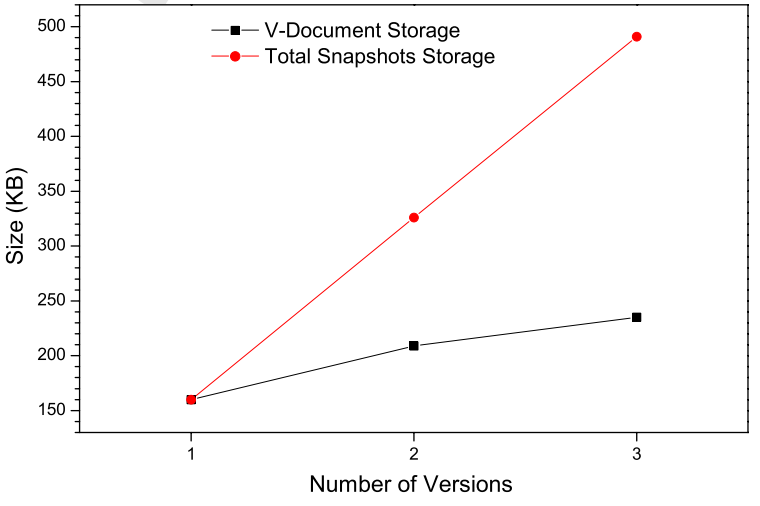

Fig. 10. XLink standards: storage used by V-Documents versus snapshots.

Please cite this article in press as: F. Wang, C. Zaniolo, Temporal queries and version management ..., Data Knowl. Eng. (2007), doi:10.1016/j.datak.2007.08.002

the changes in the document. A first possibility is to append them to this first version as shown in Fig. 11. As discussed in [28], this organization presents some advantages but destroys the clustering of the original document. A better approach consists of retaining the basic structure of Fig. 5, whereby the salary history of a given employee is stored sequentially; thus the deltas of each element are appended to the end of the element, rather than at the end of the whole document. However, as the history becomes longer, the initial clustering of employees (by department) will be lost and, as time goes by, more and more pages are needed to retrieve the employees working in a given department at a certain time (snapshot query). The usefulness-based scheme solves this problem by monitoring the *usefulness* of each page—which is defined as the percentage of currently valid objects in the page. When the page falls below the specified threshold, the objects currently alive are copied to a new page (and future deltas will also be placed in this page) Fig. 12. Reconstructing the document at a certain time only requires retrieving the pages that were useful at that time [29]. The scheme is efficient and robust [25,28]; The performance of snapshot queries on multi-version documents is basically that of queries on a single version stored in smaller pages (i.e., pages whose size is reduced by the usefulness factor). subsubsectioShredding and relational databases

A frequently used technique consists in decomposing the documents into flat tables that can be more efficiently managed by current OR DBMS. This technique is particularly effective in the case where the V-Documents actually represent an XML view of the history of an underlying relational database [56]. In [56,58], the authors showed that the shredded V-Documents can be stored and managed efficiently as a set of relational tables, whereby a separate table is used for the history of each attribute (such as salary). The usefulness-based clustering scheme discussed above and standard relational indexing techniques are then all that is needed to manage the history of our database efficiently. Complex historical queries written in XQuery can then be implemented via their equivalent XML/SQL queries on such history relations [56]. Additional refinements, such as schema history management and compression, can also be used [58].

We were able to solve the scalability problem for V-Documents representing the history of relational database tables, but the case of arbitrary XML documents is more complex and requires more sophisticated techniques, such as durable node numbers whereby information on the structure of the document is not lost upon shredding.

6.4.2. Durable node numbers

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 [62,43,52,29]. The SPaR(Sparse Preoder and Range) numbering scheme [29] uses durable node numbers (DNN, range) that can sustain frequent updates, which was first proposed in [43]. Thus the interval $[dnn(X),\ dnn(X) + range(X)]$ is associated with element X, and we represent it as [*vstart, vend*] in the V-Document (Fig. 5).

6.4.3. Complex queries

The use of DNN also facilitates the maintenance of indexes on multi-version documents. In fact, by using DNNs, efficient indexing schemes [29] and query processing algorithms [30,28] can be used to support complex queries on multi-version documents. For instance, multi-version B-Trees (MVBT) [19] indexing is used to support complex queries. The scheme [28] supports conventional and path expression queries.

```
<employee vstart="1995-01-01" vend="1996-12-31">
 <id vstart="1995-01-01" vend="1996-12-31">Bob</id>
  <name vstart="1995-01-01" vend="1996-12-31">Bob</name>
  <salary vstart="1995-01-01" vend="1995-05-31">60000</salary>
  <title vstart="1995-01-01" vend="1995-09-30" >Engineer</title>
  <deptno vstart="1995-01-01" vend="1995-09-30">d01</deptno>
<salary vstart="1995-06-01" vend="1996-12-31">70000</salary>
</employee>
<title vstart="1995-10-01" vend="1996-1-31">Sr Engineer</title>
<deptno vstart="1995-10-01" vend="1996-12-31">d02</deptno>
<title vstart="1996-02-01" vend="1996-12-31">Tech Leader</title>
```

Fig. 11. Storage organization.

Please cite this article in press as: F. Wang, C. Zaniolo, Temporal queries and version management ..., Data Knowl. Eng. (2007), doi:10.1016/j.datak.2007.08.002

UNCORRECTED PROOF

To implement such clustering/indexing techniques for efficient storage and retrieval of multi-version XML documents, the XML database has to be adapted accordingly with additional effort. For example, the durable node numbers will be assigned and indexed for query processing where additional query optimization is needed.

Therefore, our general approach to supporting historical queries using XML and XQuery can be applied to both relational databases and XML documents. But for the first case, mature technology is at hand to assure performance and scalability, while in the second case the enabling technology is still coming of age. Because of the continuous efforts by researchers and vendors, we see good progress on this front. For instance, the first-generation XML extensions for ORDBMS were often based on character large objects (CLOBs), whereby updating any element would normally require the deletion and reinsertion of the whole document. However, the latest versions of commercial systems have addressed these problems, and thus can provide for efficient incremental updating of our V-Document. The testbed applications that we studied so far include W3C documents [14], the UCLA course catalog [9], and the *CIA World Factbook [2]*. These documents are of relatively modest size (less than 10 megabytes) and the same is true for their multi-version history—a practical confirmation of the good storage efficiency of V-Documents that follows from the simple analytical model presented in Appendix 2. For documents of this size, the performance of our historical queries on both native XML databases and OR DBMS is quite satisfactory in practice. However, it is also clear that more research is needed to develop techniques whereby XML databases can achieve levels of performance comparable to relational databases. Furthermore, extensions of such techniques for historical queries represent an important facet of this ongoing research endeavor.

7. Conclusions

Preserving Web information for future uses represents a critical requirement for the modern information society. In addition to retrieval and browsing capabilities, the preservation of XML repositories and relational databases also requires the ability of supporting queries on the archived documents. At the present, these requirements are not met and await for the development of new enabling technology. Therefore in this paper, we have proposed a novel approach to the management of Web document archives and data warehouses, which, in addition to preserving the original artifacts, also supports powerful historical queries on the evolution of the documents and their contents. Therefore, we have presented simple techniques (based on the hierarchical timestamping of XML elements) and shown that said techniques can be used to represent and query temporal information in XML and that they are effective on a wide assortment of information sources ranging from textual documents to transaction-time relational databases.

The key features of the proposed approach are: (i) the evolution history of documents and databases is represented in standard XML using a temporally-grouped representation and (ii) complex temporal queries are then expressed in XQuery, without requiring temporal extensions to the XML standards. Indeed, while we relied on the use of several temporal functions to simplify our queries, similar functions can be easily added by users to their current XML/XQuery systems. Therefore, an interesting conclusion that follows from these results is that supporting temporal models and historical queries is significantly easier in an XML/XQuery-based approach than in the traditional framework of relational tables and SQL. (This observation holds even when SQL is enriched with temporal user-defined functions; in fact, even when enriched with special temporal extensions, SQL failed to gain much popularity as a temporal query language.)

After focusing on the problems of modeling and querying temporal information at the logical level, we briefly discussed various clustering and indexing approaches to achieve efficient execution of these queries. The main issues regarding performance and scalability of (i) XML databases and (ii) temporal information and queries were identified and discussed. Efficient support for temporal queries in the XML/XQuery framework is thus emerging as an area of significant research opportunities and great importance for temporal applications [28].

8. Uncited references

Q1 [7,31].

UNCORRECTED PROOF

Please cite this article in press as: F. Wang, C. Zaniolo, Temporal queries and version management ..., Data Knowl. Eng. (2007), doi:10.1016/j.datak.2007.08.002

Acknowledgements

The authors thank Shu-Yao Chien, Bertram Ludäscher, Richard Marchiano, and Vassilis Tsotras for many inspiring discussions. We also thank Emanuele Ottavi for his implementation of XChronicler. This research work was carried out as part of the ICAP (Incorporating Change Management into Archival Processes) project sponsored by the National Historical Publications and Records Commission. We also want to thank the reviewers for their insightful suggestions, which we were able to implement thoroughly, and which resulted in Q2 significant improvements to the quality of the paper.

Appendix 1. XML schemas of the sample documents

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema"
  elementFormDefault="qualified">
  <xs:element name="document">
    <xs:complexType>
      <xs:sequence>
        <xs:element ref="title"/>
        <xs:element ref="section" maxOccurs="unbounded"/>
      </xs:sequence>
    </xs:complexType>
  </xs:element>
  <xs:element name="section">
    <xs:complexType>
      <xs:sequence>
        <xs:element ref="title"/>
        <xs:element ref="subsection" maxOccurs="unbounded"/>
      </xs:sequence>
      <xs:attribute name="no" type="xs:string" use="required"/>
    </xs:complexType>
  </xs:element>
  <xs:element name="subsection" type="xs:string"/>
  <xs:element name="title" type="xs:string"/>
</xs:schema>
```

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema"
    elementFormDefault="qualified">
  <xs:element name="document">
    <xs:complexType>
      <xs:sequence>
        <xs:element ref="title"/>
        <xs:element ref="section" maxOccurs="unbounded"/>
      </xs:sequence>
      <xs:attribute name="vstart" type="xs:date" use="required"/>
      <xs:attribute name="vend" type="xs:date" use="required"/>
    </xs:complexType>
  </xs:element>
  <xs:element name="no">
    <xs:complexType>
      <xs:simpleContent>
        <xs:extension base="xs:string">
          <xs:attribute name="isAttr" type="xs:string"/>
          <xs:attribute name="vstart" type="xs:date" use="required"/>
          <xs:attribute name="vend" type="xs:date" use="required"/>
        </xs:extension>
      </xs:simpleContent>
    </xs:complexType>
  </xs:element>
  <xs:element name="section">
    <xs:complexType>
      <xs:sequence>
        <xs:element ref="no"/>
        <xs:element ref="title" maxOccurs="unbounded"/>
        <xs:element ref="subsection" maxOccurs="unbounded"/>
      </xs:sequence>
      <xs:attribute name="vstart" type="xs:date" use="required"/>
      <xs:attribute name="vend" type="xs:date" use="required"/>
    </xs:complexType>
  </xs:element>
  <xs:element name="subsection">
    <xs:complexType mixed="true">
      <xs:attribute name="vstart" type="xs:date" use="required"/>
      <xs:attribute name="vend" type="xs:date" use="required"/>
    </xs:complexType>
  </xs:element>
  <xs:element name="title">
    <xs:complexType mixed="true">
      <xs:attribute name="vstart" type="xs:date" use="required"/>
      <xs:attribute name="vend" type="xs:date" use="required"/>
    </xs:complexType>
  </xs:element>
</xs:schema>
```

Fig. 12. (a) XML schema of the sample snapshot XML document and (b) XML schema of the sample V-Document.

Appendix 2. Efficient storage

V-Documents also achieve efficient storage compared to snapshot-based versions, especially when the ratio of update on documents is small.

Sauppose the deletion ratio, insertion ratio, and update ratio are R_{del}, R_{ins}, and R_{upd}, respectively. The size of snapshot version n is S_n, and the total size of versions up to version n is $S_{1\to n}$. Here for simplification, we assume that the update will not increase the size in a snapshot version; the size of additional timestamping attributes can be ignored.

For the snapshot-based scheme, the size of S_n is as follows:

$$S_n = S_{n-1}(1 + R_{\text{ins}} - R_{\text{del}}) \tag{1}$$

The total version size up to version n is

$$V_{1\to n} = S_1 + S_2 + \cdots + S_{n-1} + S_n = S_1 \frac{(1 + R_{\text{ins}} - R_{\text{del}})^n - 1}{R_{\text{ins}} - R_{\text{del}}} \tag{2}$$

When $(R_{\text{ins}} - R_{\text{del}}) \ll 100\%$, Eq. (2) is approximately equal to

$$V_{1\to n} = S_1 \frac{1 + n(R_{\text{ins}} - R_{\text{del}}) - 1}{R_{\text{ins}} - R_{\text{del}}} = n \cdot S_1 \tag{3}$$

For V-Documents, we have

$$V_{1\to n} = V_{1\to n-1}(1 + R_{\text{ins}} + R_{\text{upd}}) \tag{4}$$

Thus,

$$V_{1\to n} = V_{1\to 1}(1 + R_{\text{ins}} + R_{\text{upd}})^{n-1} \approx S_1(1 + R_{\text{ins}} + R_{\text{upd}})^{n-1} \tag{5}$$

where $V_{1\to 1}$ is close to S_1.

When $(R_{\text{ins}} + R_{\text{upd}}) \ll 100\%$, Eq. (5) is approximately equal to

$$V_{1\to n} = S_1(1 + (n-1)(R_{\text{ins}} + R_{\text{upd}})) \tag{6}$$

When $n \gg 1$, we have

$$S_{1\to n} = n \cdot S_1 \cdot (R_{\text{ins}} + R_{\text{upd}}) \tag{7}$$

Eqs. (3) and (7) clearly show that when the update ratio is small, the V-Document has a significant storage advantage.

References

[1] Archival Information Systems (ArchIS): Publishing and Querying the Transaction-Time History of Databases in XML, <http://wis.cs.ucla.edu/projects/archis/index.html>.
[2] CIA: The World Factbook, <http://www.cia.gov/cia/publications/factbook/>.
[3] DeltaXML, <http://www.deltaxml.com/>.
[4] The Internet Archive–The Wayback Machine, <http://www.archive.org/>.
[5] Microsoft XML Diff, <http://apps.gotdotnet.com/xmltools/xmldiff/>.
[6] Oracle XML DB, <http://www.oracle.com/technology/tech/xml/xmldb/>.
[7] SQL/XML, <http://www.sqlx.org>.
[8] Tamino XML Database, <http://www.tamino.com/>.
[9] UCLA Catalog, <http://www.registrar.ucla.edu/catalog/>.
[10] UCLA ICAP Project, <http://wis.cs.ucla.edu/projects/icap/>.
[11] The Versioning Machine, <http://mith2.umd.edu/products/ver-mach/>.
[12] WebDAV, WWW Distributed Authoring and Versioning, www.ietf.org/html.charters/webdav-charter.html.
[13] X-Hive XML Server, <http://www.xhive.com/>.
[14] XML Linking Language (XLink), <http://www.w3.org/TR/xlink/>.
[15] XQuery 1.0: An XML Query Language, <http://www.w3.org/TR/xquery/>.
[16] XSL Transformations (XSLT), <http://www.w3.org/TR/xslt/>.

UNCORRECTED PROOF

Please cite this article in press as: F. Wang, C. Zaniolo, Temporal queries and version management ..., Data Knowl. Eng. (2007), doi:10.1016/j.datak.2007.08.002

[17] National Archives of Australia's Policy Statement Archiving Web Resources: A Policy for Keeping Records of Web-based Activity in the Commonwealth Government, <http://www.naa.gov.au/recordkeeping>.
[18] T. Amagasa, M. Yoshikawa, S. Uemura, A data model for temporal XML documents, in: DEXA, 2002.
[19] B. Becker, S. Gschwind, T. Ohler, B. Seeger, P. Widmayer, On optimal multiversion access structures, in: Proceedings of the Symposium on Large Spatial Databases, vol. 692, 1993.
[20] David Beech, Brom Mahbod, Generalized version control in an object-oriented database, in: ICDE, 1988, pp. 14–22.
[21] P. Buneman, S. Khanna, K. Tajima, W. Tan, Archiving scientific data, ACM Trans. Database Syst 29 (1) (2004) 2–42.
[22] Elisa Bertino, Elena Ferrai, Giovanna Guerrini, A formal temporal object-oriented data model, in: EDBT, 1996.
[23] S. Chawathe, A. Rajaraman, H. Garcia-Molina, J. Widom, Change detection in hierarchically structured information, in: SIGMOD, 1996.
[24] E. Camossi, E. Bertino, G. Guerrini, M. Mesiti, Automatic evolution of multigranular temporal objects, in: TIME, 2002.
[25] S.-Y. Chien, V.J. Tsotras, C. Zaniolo, Version Management of XML Documents, in: WebDB, 2000.
[26] S.-Y. Chien, V.J. Tsotras, C. Zaniolo, Copy-based versus edit-base version management schemes for structured documents, in: RIDE, 2001.
[27] S.-Y. Chien, V.J. Tsotras, C. Zaniolo, Efficient management of multiversion documents by object referencing, in: VLDB, 2001.
[28] S.-Y. Chien, V. Tsotras, C. Zaniolo, D. Zhang, Supporting complex queries on multiversion XML documents, in: ACM TOIT, February 2006.
[29] S.-Y. Chien, V.J. Tsotras, C. Zaniolo, D. Zhang, Storing and querying multiversion XML documents using durable node numbers, in: WISE, 2001.
[30] S.-Y. Chien, Z. Vagena, D. Zhang, V.J. Tsotras, C. Zaniolo, Efficient structural joins on indexed XML documents, in: VLDB, 2002.
[31] C.X. Chen, C. Zaniolo, Universal temporal extensions for database languages, in: ICDE, 1999.
[32] J. Chomicki, D. Toman, M.H. Böhlen, Querying ATSQL databases with temporal logic, ACM TODS 26 (2) (2001) 145–178.
[33] J. Clifford, Formal Semantics and Pragmatics for Natural Language Querying, Cambridge University Press, 1990.
[34] J. Clifford, A. Croker, F. Grandi, A. Tuzhilin, On temporal grouping, in: Proceedings of the International Workshop on Temporal Databases, 1995.
[35] C.E. Dyreson, Observing transaction-time semantics with TTXPath, in: WISE, 2001.
[36] Gregory Cobena, Serge Abiteboul, Amelie Marian, Detecting changes in XML documents, in: ICDE, 2002.
[37] Hong-Tai Chou, Won Kim, A unifying framework for version control in a CAD environment, in: VLDB, 1986, pp. 336–344.
[38] J. Jacob, A. Sachde, S. Chakravarthy, CX-DIFF: a change detection algorithm for XML content and change visualization for WebVigiL, Data Knowledge Eng 52 (2004) 209–230.
[39] D. Gao, R.T. Snodgrass, Temporal slicing in the evaluation of XML queries, in: VLDB, 2003.
[40] M. Gergatsoulis, Y. Stavrakas, Representing changes in xml documents using dimensions, in: Xsym, 2003.
[41] F. Grandi, F. Mandreoli, The valid web: an XML/XSL infrastructure for temporal management of web documents, in: ADVIS, 2000.
[42] S. Kepser. A simple proof for the turing-completeness of XSLT and XQuery, in: Extreme Markup Languages, 2004.
[43] Q. Li, B. Moon, Indexing and querying XML data for regular path expressions, in: VLDB, 2001.
[44] A. Marian et al., Change-centric management of versions in an XML warehouse, in: VLDB, 2001.
[45] S. Pal, M. Fussell, I. Dolobowsky, XML support in Microsoft SQL Server 2005, <http://msdn.microsoft.com/library/>.
[46] A.O. Mendelzon, F. Rizzolo, A. Vaisman, Indexing temporal XML documents, in: VLDB, 2004.
[47] B. Oliboni, E. Quintarelli, L. Tanca, Temporal aspects of semistructured data, in: TIME, 2001.
[48] G. Ozsoyoglu, R.T. Snodgrass, Temporal and real-time databases: a survey", IEEE Trans. Knowledge Data Eng. 7 (4) (1995) 513–532.
[49] Dimitris Papadias, Yufei Tao, Panos Kalnis, Jun Zhang, Indexing spatio-temporal data warehouses, in: ICDE, 2002.
[50] M.J. Rochkind, The source code control system, IEEE Trans. Software Eng. SE-1 (4) (1975) 364–370.
[51] R. Snodgrass, Temporal object-oriented databases: a critical comparison, in: Modern Database Systems: The Object Model, Interoperability and Beyond, Addions-Wesley/ACM Press, 1995.
[52] 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.
[53] W.F. Tichy, RCS—A System for Version Control, Software – Practice Exp. 15 (7) (1985) 637–654.
[54] Fabio Vitali, Versioning hypermedia, ACM Comput. Surveys 31 (4es) (1999) 24.
[55] F. Wang, C. Zaniolo, Preserving and querying histories of XML-published relational databases, in: ECDM, 2002.
[56] F. Wang, C. Zaniolo, Publishing and querying the histories of archived relational databases in XML, in: WISE, 2003.
[57] F. Wang, C. Zaniolo, Temporal queries in XML document archives and web warehouses, in: TIME, 2003.
[58] F. Wang, X. Zhou, C. Zaniolo, Using XML to build efficient transaction-time temporal database systems on relational databases, Technical Report 81, TimeCenter, http://www.cs.auc.dk/TimeCenter, March 2005.
[59] Y. Wang, D.J. DeWitt, J. Cai, X-Diff: a fast change detection algorithm for XML documents, in: ICDE, 2003.
[60] Jun Yang, Temporal Data Warehousing, Ph.D. Dissertation, Stanford University, 2001.
[61] C. Zaniolo, S. Ceri, C. Faloutsos, R.T. Snodgrass, V.S. Subrahmanian, R. Zicari, Advanced Database Systems, Morgan Kaufmann Publishers, 1997.
[62] C. Zhang, J.F. Naughton, D.J. DeWitt, Q. Luo, G.M. Lohman, On supporting containment queries in relational database management systems, in: SIGMOD, 2001.
[63] S. Abiteboul, S. Cluet, G. Ferran, M. Rousset, The Xyleme project, Comp. Networks 39 (3) (2002) 225–238.
[64] Y. Chen, S.K. Madria, S.S. Bhowmick, DiffXML: change detection in XML data, in: DASFAA, 2004, pp. 289–301.

Please cite this article in press as: F. Wang, C. Zaniolo, Temporal queries and version management ..., Data Knowl. Eng. (2007), doi:10.1016/j.datak.2007.08.002

UNCORRECTED PROOF