DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING

XML-based Temporal Models

by Khadija Ali and Jaroslav Pokorný

Research Report DC-2006-02

Czech Technical University in Prague
Czech Republic

www: *http://cs.felk.cvut.cz*

Department of Computer Science and Engineering
Faculty of Electrical Engineering
Czech Technical University in Prague

XML-based Temporal Models

Research Report DC-2006-02

by Khadija Ali and Jaroslav Pokorný

June 2006

Contact address:

Department of Computer Science
and Engineering
Faculty of Electrical Engineering
Czech Technical University in Prague
Karlovo nám. 13
CZ – 121 35 Praha 2
Czech Republic

Phone: (+420) 224 357 470
Fax: (+420) 224 923 325
e-mail: *office@cs.felk.cvut.cz*
www: *http://cs.felk.cvut.cz*

Dean's Office:

Faculty of Electrical Engineering
Czech Technical University in Prague
Technická 2
CZ – 166 27 Praha 6
Czech Republic

Phone: (+420) 224 352 016
Fax: (+420) 224 310 784
www: *http://www.fel.cvut.cz*

The research of J. Pokorný was in part supported by grant 1ET100300419 "Intelligent Models, Algorithms, Methods and Tools for the Semantic Web Realisation" of the Information Society Program -Thematic Program II of the National Research Program of the Czech Republic.

XML-based Temporal Models
by Khadija Ali and Jaroslav Pokorný
Published by Department of Computer Science and Engineering
Czech Technical University in Prague
Karlovo náměstí 13, 121 35 Praha 2, Czech Republic
E-mail: office@cs.felk.cvut.cz Phone: (+420) 224 357 470
Printed by Nakladatelství ČVUT, Thákurova 1, 160 41 Praha 6, Czech Republic

Abstract

Much research work has recently focused on the problem of representing historical information in XML. This report describes a number of temporal XML data models and provides their comparison according to the following properties: time dimension (valid time, transaction time), support of temporal elements and attributes, querying possibilities, association to XML Schema/DTD, and influence on XML syntax.

Keywords
XML, temporal XML data model, bitemporal XML data model, XQuery, versioning XML documents, transaction time, valid time

Contents

Chapter 1

Introduction

Recently, the amount of data available in XML [12] has been rapidly increasing. In context of databases, XML is also a new database model serving a powerful tool for approaching semistructured data. Similarly to relational or object-relational models in the past, database practice with XML started to change also towards using time in some applications.

Much research work has recently focused on adding temporal features to XML, i.e. to take into account change, versioning, evolution and also explicit temporal aspects like, e.g., the problem of representing historical information in XML. Based on similar approaches well-known from the field of temporal relational databases [8], many their ideas have been transformed into the world of hierarchical structures of XML documents. In [6] a new <valid> mark-up tag for XML/HTML document is proposed to support valid time on the web, thus temporal visualization can be implemented on web browsers with XSL. In [5] a dimension-based model is proposed to represent changes in XML documents. The model is capable to represent changes not only in an XML document but to the changes corresponding XML schema as well, however how to support queries is not discussed. There are several other temporal dimensions that have been also mentioned in the literature in relation to XML. In [7] a publication time and efficiency time in the context of legal documents are proposed. In [4] the XPath data model is extended to support transaction time. In [11] the XPath data model and query language is extended to include valid time. The XPath is extended with an axis to access valid time of nodes. In [3] an archiving technique for data is presented, in which elements can be uniquely identified and retrieved by their logical keys. The elements have timestamps only if they are different from the parent node; however support for queries is not discussed. In [9] an approach of representing XML document versions is proposed by adding two extra attributes, namely *vstart* and *vend*, representing the time interval for which an element version is valid. The model can also support powerful temporal queries expressed in XQuery without requiring the introduction of new constructs in the language.

The report is organized as follows. In Chapter 2 we describe an XML-based bitemporal data model (XBIT). This general technique is applied in an XML-based model for versioned documents in Chapter 3. In Chapter 4 we describe a temporal XML data model which is able to manage the dynamics of norms in time. We introduce a valid-time XPath data model in Chapter 5. The model adds valid time support to XPath.

In Chapter 6 we present a brief overview of the Diff-based approach for archiving data, which will be compared with a key-based approach in Chapter 7. A dimension-based model is introduced in Chapter 8. Finally, in Chapter 9 we summarize all the mentioned models. We briefly analyze their characteristics and subsequently we express our point of view.

Chapter 2

XBIT- an XML-based Bitemporal Data Model

The used technique is general and can be applied to historical representation of relational data (as it will be shown in the next subsections), and versions management in archives (as it will be shown in chapter 3. The approach is based on temporally-grouped data model[1], which is compatible perfectly with the hierarchical structure of XML documents. Temporal XML document is represented by adding two extra attributes, namely *vstart* and *vend*, representing the time interval for which an element is valid. The model can also support powerful temporal queries expressed in XQuery without requiring the introduction of new constructs in the language [10].

2.1 Data Modelling

Consider Figure 2.1, which shows a bitemporal history of employees, as it would be viewed in a traditional relational representation, where each tuple is time stamped with a valid time interval, and a transaction time interval. The valid time end can be set to *now*, while the transaction time end can be set to *UC* (*until changed*). This table representation of employee entities is temporally-ungrouped. It has several drawbacks:

1. Redundant information is preserved between tuples.
2. Temporal queries need to coalesce frequently tuples, which is the source of complications in temporal query languages.

These problems can be overcome using a temporally grouped representation, as shown in Figure 2.2. For instance, the last square of the last column in Figure 2.2 says that Bob's salary is 85000 from "2000-01-01" till now. This fact is recorded in the system in "1999-09-01". In temporally grouped representation the time stamped history of each attribute is grouped under the attribute. XBIT supports a temporally grouped

[1]In [8] the relational data models are classified as two main categories: *temporally- ungrouped models* and *temporally grouped* data models. *Temporally grouped* data model can be expressed by relations in non-first-normal-form model or attribute time stamping, in which the domain of each attribute is extended to include the temporal dimension.

NAME	TITLE	DEPT	SALARY	VALID TIME	TRANSACTION TIME
Bob	Sr Engineer	RD	70000	1998-01-01:NOW	1997-09-01: 1999-08-31
Bob	Sr Engineer	RD	70000	1998-01-01:1999-12-31	1999-09-01:UC
Bob	Sr Engineer	RD	85000	2000-01-01:NOW	1999-09-01:UC

Figure 2.1: Bitemporal history of Employees.

Name	Title	Dept	Salary
v:1998-01-01 v:now	v:1998-01-01 v:now	v:1998-01-01 v:now	v:1998-01-01 v:now 70000 t:1997-09-01 t:1999-08-31
			v:1998-01-01 v:1999-12-31 70000 t:1999-09-01 t:UC
Bob	Sr Engineer	RD	v:2000-01-01 v:now 85000
t:1997-09-01 t:UC	t:1997-09-01 t:UC	t:1997-09-01 t:UC	t:1999-09-01 t:UC

Figure 2.2: Temporally grouped bitemporal history of employees.

representation by coalescing attributes's histories on both transaction and valid time. For bitemporal histories, coalescing is done when two tuples are value-equivalent (the values of their corresponding relational attributes are the same. For instance, the first and second tuple in Figure 2.1 are value-equivalent. In both tuples the attributes: name, title, dept, and salary have the values: Bob, Sr-Engineer, RD, and 70000 respectively) and

1. their valid time intervals are the same and the transaction time intervals meet or overlap,
2. or the transaction time intervals are the same and the valid time intervals meet or overlap.

This operation is repeated until no tuples satisfy these conditions. For example, in Figure 2.1, to group the history of titles with values "Sr-Engineer" in the three tuples, i.e., (title, valid-time, transaction-time), the last two transaction time intervals are the same, and the last two valid time intervals meet, so they are coalesced as (Sr-Engineer, 1998-01-01: now, 1999-09-01: UC). This one again has the same valid time intervals as the previous one: (Sr-Engineer, 1998-01-01: now, 1997-09-01:1999-08-31). Thus they are coalesced finally as (Sr-Engineer, 1998-01-01: now, 1997-09-01: UC) as shown in the second column of Figure 2.2. The bitemporal history of employees is represented in XBIT as an XML document (BH-document), as shown in Figure 2.3, where:

1. Each employee entity is represented as an employee element in the BH-document.
2. Table attributes are represented as employee element's child elements.
3. Each element in the BH-document is assigned two pairs of attributes; *tstart* and *tend* to represent the inclusive transaction time interval (*tend* can be to *uc* (*until changed*)), *vstart* and *vend* represent the inclusive valid time interval (*vend* can be set to *now*) to denote the ever-increasing current date.
4. There is a time covering constraint that the time interval of a parent node always covers that of its child nodes.
5. Although valid time and transaction time are generally independent, for the sake of illustration, it is assumed that employee's promotions are scheduled and entered in the database four months before they occur.

```
<employees vstart="1998-01-01" vend="now" tstart="1997-09-01"
                tend="UC">
      <employee vstart="1998-01-01" vend="now" tstart="1997-09-
              01"tend="UC">
      <name vstart="1998-01-01" vend="now" tstart="1997-09-01"
              tend="UC">Bob</name>
      <title vstart="1998-01-01" vend="now" tstart="1997-09-01"
              tend="UC">Sr Engineer</title>
      <dept vstart=""1998-01-01" vend="now" tstart="1997-09-01"
              tend="UC">RD</dept>
      <salary vstart="1998-01-01" vend="now"  tstart="1997-09-01"
          tend="1999-08-31">70000</salary>
      <salary vstart="1998-01-01" vend="1999-12-31"
              tstart="1999-09-01" tend="UC">70000
      </salary><salary vstart="2000-01-01" vend="now"
          tstart="1999-09-01" tend="UC">85000</salary>
    </employee>
</employees>
```

Figure 2.3: XML representation of the bitemporal history of employees (BH-document).

2.2 Bitemporal Queries with XQuery

The XBIT supports powerful temporal queries, expressed in XQuery without introducing new constructs in the language.

Considering the previous example (BH-document):

- It is possible to retrieve the history of any element (for instance salary's history of a specific employee).

```
for $s in doc("emps.xml")/employees/employee[name ="Bob"]/salary
return $s
```

- Snapshot queries for instance the salaries in a specific valid time and transaction time. The following query retrieves the average salary on 1999-05-01, according to what was known on that time.

```
let $s := doc("emps.xml")/employees/employee/salary
where tstart($s)<= "1999-05-01" and tend($s) >= "1999-05-01" and
vstart($s) <= "1999-05-01" and vend($s) >= "1999-05-01" return
avg($s)
```

  Here *tstart*(), *tend*(), *vstart*(), and *vend*() are user-defined functions that get the starting date and ending date of an element 's transaction-time and valid-time, respectively.

- The query may take a transaction time snapshot and valid time slicing of any element, for instance to retrieve employees whose salaries (according to our current information) did not change between 1999-01-01 and 2000-01-01:

```
let $s := doc("emps.xml")/employees/employee/salary
where tstart($s) <= current-date() and tend($s) >= current-date()
and vstart($s) <= "1999-01-01" and vend($s) >= "2000-01-01"
return $s/..
```

2.3 The Modifications

Modification in XBIT can be seen as the combination of modification on valid time and transaction time history. XBIT will automatically coalesce on both valid time and transaction time. XBIT modifications can be classified as of three types: insert, delete, and update.

Insert

The added element and its child elements are time stamped with valid time interval as (*vstart*, *now*) and a transaction time interval as (*current date*, *UC*).

Delete

When an element is removed at *time*, the value of *vend* is changed to *time*−1. The deletions with a valid time on a node are propagated downward to its children to satisfy the covering constraint. The ending transaction time *tend* of the deleted element and it is current children changed to current date.

Update

Update can be seen as a delete followed by an insert. The updates can be on values or valid time, for value update, propagation is not needed; for valid time update, it is needed to downward update the nodes's children's valid time.

Chapter 3

An XML-based Model for Versioned Documents

An efficient technique for managing multiversion document histories is used by storing the successive versions of a document in an incremental fashion, and supporting powerful temporal queries on such documents expressed in XQuery without requiring the introduction of new constructs in the language. XML document is represented by adding two extra attributes, namely *vstart* and *vend* representing the time interval for which this elements version is valid [9].

3.1 Data Modeling

The successive versions of a document are represented as an XML document called a V-Document. Figure 3.1 shows a sample of versioned XML document, while Figure 3.2 shows the XML representation of this document. In a V-Document the following conditions hold:

1. Each element is assigned two attributes *vstart* and *vend* which represent the valid versions intervals (inclusively) of the element. In general, *vstart* and *vend* can be versions numbers or timestamps.
2. *vstart* represents the initial version when the element is first added to the XML document, vend represents the last version in which such an element is valid. After the vend version, the element is either removed or changed.
3. The value of *vend* can be set to *now* to denote the ever increasing current version number.
4. Elements containing attributes are represented by a sub-element denoted by a special flag attribute *isAt*. For instance if the employee element contains the attribute *empn* then this is represented as:
   `<empno isAtt="yes" vstart="1999-01-01" vend="now">` e_1 `</empno>`
5. The version interval of an element always contains those of its descendents

```
<<document>            <!--comment: This is version 1 -->
      <chapter no="1">
          <title> Introduction</title>
              <section>Background</section>
              <section>Motivation</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>
```

Figure 3.1: Sample versioned XML Documents.

```
<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="2002-01-01" 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="now">XML storage
      </section>
    </chapter>
</document>
```

Figure 3.2: XML representation of versioned document (V-document).

3.2 Data Modification

Three primitive change operations are considered, delete, insert, and update. The following is the effect of performing each operation:

Update

When an element is updated

1. a new element with the same name will be appended immediately after the original one; the attributes *vstart* and *vend* of this new element are set to the current version number and the special symbol "*now*, respectively.
2. the *vend* attribute of the old element is set to the last version before it was changed.

Insert

When a new element is inserted, this 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 *vend* attribute is set to the last version where the element was valid.

3.3 DTD of the V-document

The DTD of the versioned XML document in Figure 3.1 is shown in Figure 3.3. The DTD of V-document in Figure 3.2, can be automatically generated as it is shown in Figure 3.4. Two new attributes *vstart* and *vend* are added to each element; an attribute of an element will be converted as a child element.

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

Figure 3.3: DTD of the versioned document in Figure 3.1.

3.4 Queries with XQuery

Due to the temporally grouped features of the model, it is possible to express powerful temporal queries. Consider the previous example shown in Figure 3.2:

1. Retrieve the version of the document on 2002-01-03.

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

```
 <!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 3.4: DTD of the V-document in Figure 3.2.

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

2. Find titles that did not change for more 2 consecutive years.

```
for  $title in document ("V-Document.xml")/document/chapter/title
let $duration:= subtract-dates ($title/@vend,$title/@vstart)
where dayTimeDuration-greater-than($duration, "p730D")
return $title
```

Here, p730D is the duration constant of 730 days in XQuery.

3. Find chapters in which the title did not change until a new section "History" was added.

```
for $ch in document ("V-Docment.xml")/document/chapter
let $title:= $ch/title[1]
let $sec:= $ch/section[section="History"]
where not empty($title) and not empty($sec) and $title/@vend =
$sec/@vstart
return $ch
```

Chapter 4

An XML-based Temporal Data Model for the Management of Versioned Normative Texts

There are several dimensions that have been mentioned in the literature in relation to XML. In this model [7], four dimensions (publication, validity, efficacy, and transaction times) are used in the context of legal documents. They are used to represent the evolution of norms in time and their resulting versioning. For the management of norms, three basic operators are defined; one for the reconstruction of a consistent temporal version and the other two for the management of textual and temporal changes.

4.1 Modelling Time and Norms

The model is based on a hierarchical organization of normative texts (i.e. legal norms can be based on a contents-section-article-paragraph hierarchy) which is particularly suitable to XML encoding. Four temporal dimensions are used to represent correctly the evolution of norms in time and their resulting versioning:

1. Publication time: it is the time of publication of norms in an official journal.
2. Validity time: it represents the time the norm is in force (the time the norm actually belongs to the regulations in the real world).
3. Efficacy time: usually corresponds to the validity of norms, but sometimes the cancelled norm continues to be applicable to a limited number of cases.
4. Transaction time: It is the time the norms is stored in the computer system.

All the above dimensions are independent. An alternative XML encoding scheme has been developed for normative text based on an XML-schema which allows the introduction of time stamping metadata at each level of the document structure which is a subject to change. Figure 4.1 depicts the XML-schema for the representation of norms in time where:

- "R" and "O" near attribute names stand for required and optional, respectively.

- The meta-level of normative texts is rooted by the *norm* element.
- The publication date is the property of the overall document and thus it has been modeled as an attribute associated to the outermost element.
- At each level of the *contents-section-article-paragraph* hierarchy it is possible to represent different versions (by means of *ver* element).
- The *ver* element is assigned associated timestamps (the three other dimensions).
- The active norm reference *an_ref* is the identifier of the modifying norm whose enforcement caused the versioning.

As an example, a fragment of a multiversion XML document, is shown in Figure 4.2; the "l247/1999" law concerns the cereal importation, it has been published on 1999/12/15, and recorded in the system on 2000-01-10, it has been valid since 2000-01-01. Paragraph 2 of chapter 1 article 1, has been modified by "LD135/2000", in force since 2000/6/1 (modification recorded on 2000/6/10)

4.2 Data Manipulation

The model is provided with three basic operators:

- O1(*vt, et, tt*) for the reconstruction of a consistent temporal version. The document is reconstructed by selecting -at each level of the hierarchy- the text making up the desired version (if it exists), *vt, et, tt* are the valid time, efficacy time, and transaction time, respectively.
- O2(*en, vts, vte, ets, ete, txt, an*) for the management of textual changes. It requires the name of the element to be substituted *en*, (*vts, vte*) and (*ets, ete*) are the validity and efficacy timestamps to be assigned to the new version, (*txt*) is the new text and, *an* is a reference to the active (i.e. the modifying) norm.
- O3(*en, vt, et, vts, vte, ets, ete, an*) for the modification of an element 's timestamps (the validity and efficacy timestamps). The new transaction time timestamps are automatically generated by the system where the transaction time start and transaction time end are *now* and *undefined* respectively.

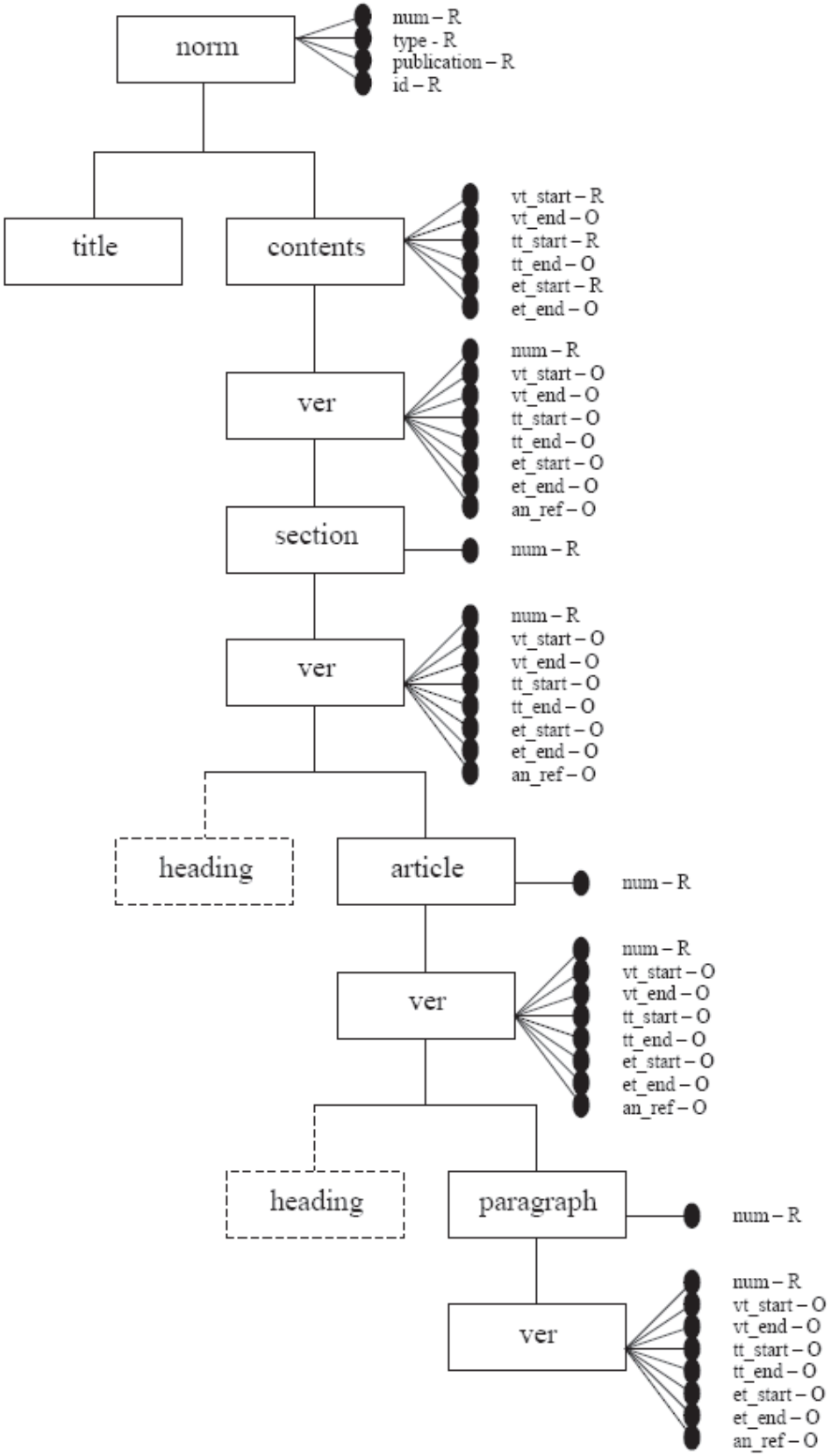

Figure 4.1: The XML schema for the representation of norms in time.

```
<norm num="247" type="law" publication="1999-12-15"
          id="l247/1999">
<title> Cereals importation </title>
<contents vt_start="2000-01-01" et_start="2000-01-01"
          tt_start="2000-01-10">
      <ver num="1">
          <section num="1">
              <ver num="1" <heading> import from communitarian
                                countries</heading>
                  <article num="1">
                      <ver num="1"> <heading> import from
                                          spain</heading>
                      <paragraph num="1">
                            <ver num="1"> Sec. 1 Art. 1 par. 1
                                 (unmodified)</ver>
                      </paragraph><paragraph num="2">
                      <ver num="1.1" vt_start="2000-01-01"
                            et_start="2000-01-01" tt_start="2000-
                            01-10" tt_end="2000-06-10"
                            an_ref="LD135/2000"> Sec. 1. Art. 1
                            par2 before modification </ver>
                      <ver num="1.2" vt_start="2000-01-01"
                            vt_end="2000-06-01" et_start="2000-
                            01-01" tt_start="2000-06-10"
                            an_ref="LD135/2000"> Sec. 1 Art.
                            1par. 2 before modification </ver>
                      <ver num="2" vt_start="2000-06-01"
                            et_start="2000-06-01" tt_start="2000-
                            06-10" an_ref="LD135/2000"> Sec. 1
                            Art1 par. 2 after modification </ver>
                      </paragraph>
                      </ver>
                  </article>
              </ver>
          </section>
      </ver>
</contents>
</norm>
```

Figure 4.2: A fragment of a multiversion XML document.

Chapter 5

A Valid Time XPath Data Model

The XPath data model and query language is extended to include valid time; the XPath is extended with an axis to access valid time of nodes. Before describing the valid time XPath model in [11], we refer briefly to the XPath data model.

5.1 The XPath Data Model

XPath is a language for specifying locations within an XML document. The XPath data model is commonly assumed to be an ordered tree. The tree represents the nesting of elements within the document, with elements corresponding to nodes, and element content comprising the children for each node. The children for a node are ordered based on their physical position within the document.

Definition 1: The *XPath data model* D for a well formed XML document X is a four-tuple $D(X) = (r, V, E, I)$ where,

- V is a set of nodes of the form (i, v) where v is the node identifier and i is an ordinal number such that for all $(i, v), (j, w) \in V$, v starts before w in the text of X if and only if $i < j$.
- E is a set of edges of the form (v, w) where $v, w \in V$. Edge (v, w) means that v is a parent of w. In terms of X, it represents that w is in the immediate content of v; since v's children are the immediate contents of v in X. There is an implied ordering among the edges; edge (v, y) is before the edge (v, z) if $y < z$ in the node ordering.
- The graph (V, E) forms a tree.
- I is the information set function which maps a node identifier to an *information set*[2].
- $r \in V$ is a special node called the *root*; r is the data model root rather than the document root. The document root is the first element node of the document.

[2]An *information set* is a collection of properties that are generated during parsing of the document. For example, an element node has the following properties: *Value* (the element's name), *Type* (element), and *Attributes* (a set of name-value pairs, in XPath, attributes are unordered).

5.2 Adding Valid Time to XPath

A node is valid at a point of time or an interval of time. But more generally, a node could be valid at several points and (or) intervals of time.

Definition 2: The *valid time* of a node in an XML document is represented as a list of time constants, $[t_1, t_2, \ldots, t_n]$, where each t_i $(i = 1, \ldots, n)$ represents a time constant when the node is valid.

- Each time constant $t = (b_i, e_i)$ is either a time interval or a time point.
- All and only the time constants when the node is valid are included in this list.
- Any two time constants do not overlap, no matter if they are time intervals or time points, i.e., $(b_i, e_i) \cap (b_j, e_j) = \emptyset$, for all $i \neq j$, $1 \leq i, j \leq n$.
- These time constants are ordered by the valid time contained in each of them, i.e., $b_{i+1} \leq e_i$, $(i = 1, \ldots, n-1)$.

For example, time intervals (1,3) and (2,4) can not be ordered; neither can time interval (1,3) and time point (2,2). Order is important in an XML document.

Definition 3: The *valid time XPath data model* D_{VT} for a well-formed XML document X is a four-tuple $D_{VT}(X) = (r, V, E, I)$, where

- V is a set of nodes of the form (i, v, t) where v is the node identifier, i is its ordinal number and t is a valid time such that the following two conditions hold:
  1. For all $(i, v, t), (j, w, s) \in V$, v starts before w in the text of X if and only if $i < j$.
  2. For all $(i, v, t) \in V$ and its children $(i_1, v_1, t_1), (i_2, v_2, t_2), \ldots, (i_n, v_n, t_n) \in V$, $t_1 \cup t_2 \cup \ldots t_n \subseteq t$.
- E, I, and r are the same as in the non-temporal XPath data model.

From definition 3, we infer that in the valid time XPath data model a list of disjoint intervals or instants that represents the valid time is added to each node, where:

- Every node of the tree structure of a well-formed XML document is associated with the valid time that represents when the node is valid, no node can exist at a valid time when its parent node is not valid.
- The valid time of any node is a superset of the union of the valid times of all its children as well as all its descendents. The valid time of the root node should be a superset of the union of the valid times of all nodes in the document.
- The valid time of an edge is determined by the valid time of the nodes at the edge's two ends (if both nodes are valid, an edge can exist). The valid time of the edge is result of $t_1 \cap t_2$, where t_1 and t_2 are the valid times of the edge's two ends.

As an example, consider the XML document "bib.xml" in Figure 5.1. Figure 5.2 depicts its data model. For brevity, not all the information is included in this figure. Each node is represented by its corresponding ordinal number beside its value. Each edge is represented as a line. For instance, the first edge is represented by a line between the first node and second node, which have the values "root" and "db", respectively. The information in the data model of "bib.xml" is sketched below.

$$
\begin{aligned}
V &= ((0,\&0),(1,\&1),\ldots,(21,\&21)) \\
E &= ((\&0,\&1),(\&1,\&2),(\&1,\&12),\ldots,(\&20,\&21)) \\
I &= (\&0,(\text{Value=root, Type=root, Attributes=“”})), \\
&\quad (\&1,(\text{Value=db, Type=element, Attributes=“”})), \\
&\quad (\&2,(\text{Value=publisher, Type=element, Attributes=“”})), \\
&\quad (\&3,(\text{Value=name, Type=element, Attributes=“”})), \\
&\quad (\&4,(\text{Value=ABC, Type=text, Attributes=“”})), \\
&\quad \ldots \\
&\quad (\&21,(\text{Value=29.99, Type=text, Attributes=“”}))) \\
r &= (0,\&0)
\end{aligned}
$$

Figure 5.3 shows the valid times of the book elements. The node set V containing the two book nodes is:

$$
\begin{aligned}
V &= ((0,\&0,t_0),\ldots,(5,\&5,t_5),\ldots,(15,\&15,t_{15}),\ldots) \\
t_5 &= [(\text{“Jan 31,1999”, now})] \\
t_{15} &= [(\text{“Jan 31,2000”, “Dec 31,2000”}),(\text{“Jan 1,2001”, now})]
\end{aligned}
$$

```
<db>
      <publisher>
          <name>ABC</name>
          <book>
              <isbn>1234</isbn>
              <title>book1</title>
              <price>19.99</price>
          </book>
      </publisher>
      <publisher>
          <name>XYZ</name>
          <book>
              <isbn>2345</isbn>
              <title>book2</title>
              <price>29.99</price>
          </book>
      </publisher>
</db>
```

Figure 5.1: The XML document "bib.xml".

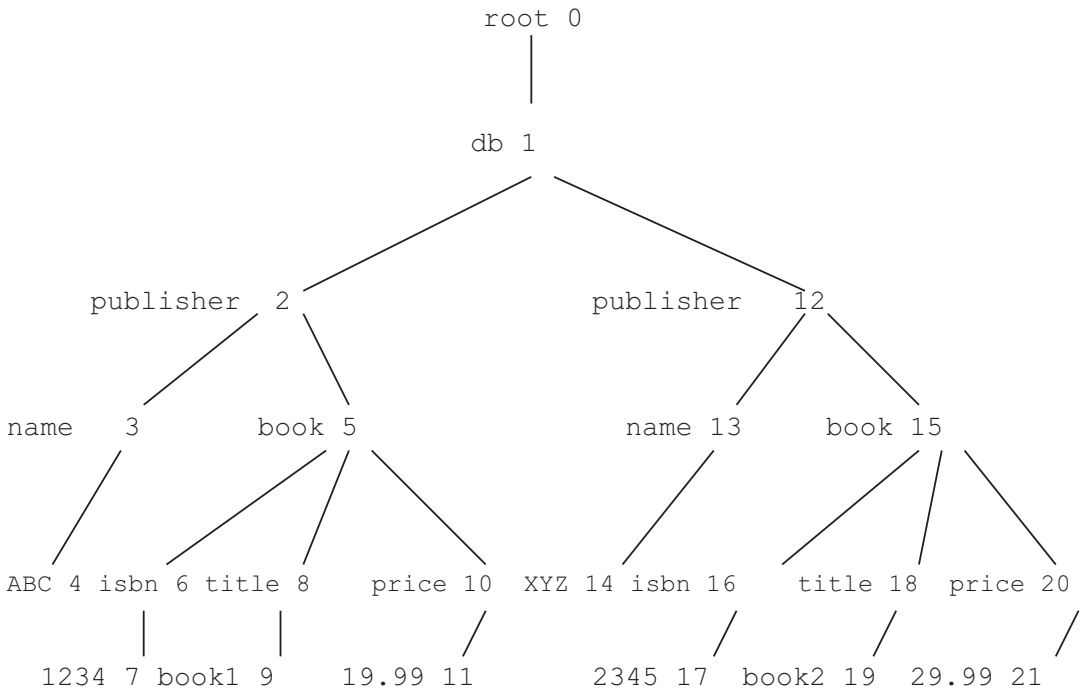

Figure 5.2: The data model for "bib.xml".

Book (isbn)	Valid time
1234	[("Jan 31,1999", now)]
2345	[("Jan 31,2000", "Dec 31, 2000"), ("Jan 1, 2001", now)]

Figure 5.3: Valid time information for book elements in "bib.xml".

5 and 15 are ordinal numbers representing the nodes positions in the tree structure-based document, &5 and &15 represent the nodes identifiers, t_5 and t_{15} are valid times of the two nodes.

5.3 Querying Valid Time

The XPath is extended with an axis to locate the valid time of a node. Each node in an XML document has a corresponding valid time view containing its valid time information.

Definition 4: A valid time list can be viewed as an XML document. Let v be a node in the data model tree of an XML document. The *valid time view* V is a mapping from the valid time for v to an XML data model X denoted $V(v) = X$.

Each time in the valid time list is denoted as a <time> element. The content of the <time> is unique to the view. Figure 5.4 shows the valid time view of the first book element in "bib.xml" in the commonly used Gregorian calendar; "year", "month", and "day" element nodes are nested under "begin" and "end" of each view.

A valid time axis is added to the query language to retrieve nodes in a view of the valid time for a node. The valid time can be viewed as an XML document in any calendar that the user has defined. The commonly used calendar is Gregorian calendar; however there are other calendars that are widely used by people in different regions.

Definition 5: The *valid time axis* selects the list of nodes that forms a document-order traversal of the *valid time view*.

By the definition 5, the nodes in the valid time axis are ordered according to the document order traversal of the valid time view. The valid time axis of a node contains the valid time information of the node as if it had originated from an XML document (a document order refers to the standard document order as it is specified in Infoset).

Since the <time> elements in the valid time view are ordered by the actual time they represent, these <time> elements selected by the valid time axis are also in this order.

Example 1: Below are some simple examples of using the valid time axis to query within the default view of the valid time.

`v/valid`	specifies the valid time axis of the node v.
`v/valid::day`	selects all the day nodes in the axis.
`v/valid::time[2]`	selects the second time node in the axis .
`v/valid(''Gregorian'')`	specifies that the calendar to use in the valid time axis of v is "Gregorian".

```
<validtime>
      <time>
           <begin>
                <day>31</day>
                <month>Jan</month>
                <year>1999</year>
           </begin>
           <end>
                <day>now</day>
                <month>now</month>
                <year>now</year>
           </end>
      </time>
</validtime>
```

Figure 5.4: The valid time view of the first book element in "bib.xml" in Gregorian calendar.

Chapter 6

Diff-based Approach

Before describing the XML-based model for archiving data in detail in Chapter 7, we first describe the main idea behind another approach for archiving data (Diff-based approach). The Diff-based approach in [3] stores the latest version together with all forward completed changes between successive versions.

6.1 Data Modeling

- The approach keeps a record of changes – a "delta" – between every pairs of consecutive versions.
- Diff algorithms are used to compute deltas.
- The latest version is stored together with all forward completed deltas – changes between successive versions – that can allow one to get to an earlier version by inverting deltas on the latest version.

Example 2: Suppose we have a database containing data of two genes, the data was corrected as shown in Figure 6.1. The diff output in Figure 6.2 explains the change as genes changing their names and id numbers. Figure 6.2 says that lines 2, 3 of version 1 should be replaced with

```
<id>2953</id> and <name>ACV2</name>
```

i.e., the gene `GRTM` changed its `id` to `2953`, and also changed its `name` to $\texttt{ACV}_2$. Similarly, the gene $\texttt{ACV}_2$ changed its `id` to `6230`, and also changed its name to `GRTM`.

The approach is efficient if we are only interested in retrieving an entire (past) version from a diff-based repository. Finding the temporal history of an element in the database may require performing some complicated analysis on diff scripts.

Version 1

```
<gene>
      <id>6230</id>
      <name>GRTM</name>
      <seq>GTCG...</seq>
      <pos>11A52</pos>
</gene>

<gene>
      <id>2953</id>
      <name>ACV2</name>
      <seq>AGTT...</seq>
      <pos>08A96</pos>
</gene>
```

Version 2

```
<gene>
      <id>2953</id>
      <name>ACV2</name>
      <seq>GTCG...</seq>
      <pos>11A52</pos>
</gene>

<gene>
      <id>6230</id>
      <name>GRTM</name>
      <seq>AGTT...</seq>
      <pos>08A96</pos>
</gene>
```

Figure 6.1: Two versions of gene elements.

```
2,3c
      <id>2953</id>
      <name>ACV2</name

8,9c
      <id>6230</id>
      <name>GRTM</name>
```

Figure 6.2: Output of Diff.

Chapter 7

XML-based Model for Archiving Data

In this archiving technique, the elements can be uniquely identified and retrieved by their logical keys, elements have timestamps only if they are different from the parent node; however support for queries is not discussed in [3].

The archiving technique used in this model stem from the requirements and properties of scientific databases. They are of course, applicable to data in other domains, but they are especially appropriate to scientific data for a variety of reasons:

1. Scientific data is inserted mostly and the transaction rate is low.
2. Much scientific data is kept in well-organized hierarchical data format and is naturally converted into XML. This hierarchically structured data usually has a key structure providing a canonical identification for every part of the document.

7.1 Data Modeling

Key-based approach is used for identifying the correspondence and changes between two given versions based on keys. In contrast to diff-based approach which keeps changes made to text documents as a sequence of delta, the Key-based approach can preserve semantic continuity of each data element in the archive. An element may appear in many versions whose occurrences are identified by using the key structure and store it only once in the merged hierarchy.

Definition 6: A *key* is a pair $(Q, \{P_1, \ldots, P_k\})$ where Q and P_i, $i \in [1, k]$ are path expressions in a syntax like XPath. Informally, Q identifies a target set of nodes reachable from some context node and this target set of nodes satisfy the key constraints given by the key paths, P_i, $i \in [1, k]$.

Example 3: Consider the following XML document

```
<DB>
      <A> <B>1</B> <C>1</C> </A>
      <A> <B>1</B> <C>2</C> </A>
</DB>
```

The document satisfies the key(`/DB/A,{C}`) but it does not satisfy the key (`/DB/A,B`) since both `A` elements have the same key path value, i.e., <B>1</B>

Example 4: The key (`/book,{isbn}`) means every `book` child of the root must have a unique `isbn` child, and if two such `book` nodes have the same value for their `isbn` children, then they are the same node. The key (`/db,(dept,{name})`) refers to that every `dept` node within a `db` node can be uniquely identified by the contents of its `name` sub-element.

This archiving technique requires that all versions of the database must conform to the same key structure and the same schema as well. Frontier node is the deepest possible keyed node. It is assumed that every node does not occur beneath frontier node is keyed (we refer to the frontier nodes with examples in Example 5).

There are nodes that can not be uniquely identified by their path or any sub elements, such nodes occur directly under a frontier node (we refer to the non-keyed nodes with examples in Example 5). The conventional Diff-approach is applied in this case.

Example 5: Consider a company database containing information about its employees and the company address.

- Every employee can be uniquely identified by his employee `id`, i.e. `id` is the key for employees.
- Each employee also has one name, at most one salary value `sal`, and optionally one telephone number `tel`.

Figure 7.1 shows a sequence of versions of the company database, while Figure 7.2 shows version 3 with key information annotated on nodes. Observe that emp nodes are annotated with their key values, e.g. `emp(id=1)`. We omit the `id` subelement of emp because they are already stored as annotations.

The key specification for the company database has the frontier paths:

```
/db/address
/db/emp/id
/db/emp/name
/db/emp/sal
db/emp/tel.
```

`name` is a frontier node but `emp` is not. If there are `first-name` and `last-name` nodes under name nodes, then these sub-elements are non-keyed nodes beyond the frontier node name.

7.2 Versions Merging

All the versions are merged into one hierarchy where an element appearing in multiple versions is stored only once along with a timestamp. Before describing the algorithm Nested merge in detail below, we first describe its main idea which is used for versions merging. The main idea behind nested merge is:

- Recursively to merge nodes in D (incoming version) to nodes in A (the archive) that have the same key value, starting from the root.
- When a node y from D is merged with a node x from A, the time stamp of x is augmented with i (the new version number). The sub-trees of nodes x and y are then recursively merged together.
- Nodes in D that do not have corresponding nodes in A are simply added to A with the new version number as its time stamp.
- Nodes in A that no longer exist in the current version D will have their timestamps terminated appropriately, i.e., these nodes do not contain timestamp i.

Before describing the algorithm, we refer to some notes or assumptions concerned it:

- A node x is value equal to a node y, denoted as $x =_v y$ if they agree on their value. The value of a node consists of its tag name and two things: (1) a possibly empty list of values of its children nodes according to the document order, and (2) a possibly empty set of values of its attributes.
- The archive A contains versions 1 through $i-1$ and D is the new version (version i). The archive A contains a single root node $r_{A.}$ for any node x in A, let time(x) denote the timestamps annotated on node x. r_D denotes the virtual root of D.
- label(x) = label(y) refers to that: The labels of the two nodes are equal, i.e., the corresponding tag names are identical and key values are the same.
- A and D are annotated with keys.
- The algorithm is invoked with the following arguments: Nested-merge $(x, y, \{\})$. The last argument contains an inherited timestamp, initially empty.

Algorithm Nested-Merge (x, y, T)

if time(x) exists **then** add i to time(x);
let T be time(x)
if y is a frontier node **then**
  if every node in children(x) is not a timestamp node **then**
    if $x \neq vy$ **then**
      create timestamp node t_1 (i.e., < T t="$T - \{i\}$">) and attach children (x) to t_1;
      create timestamp node t_2 (i.e., < T t="i">) and attach children(y) to t_2;
      attach t_1 and t_2 as children nodes of x;
else

if there exists a node x' in children(x) such that children(x') = v children (y) **then**
add i to time (x').
else
create timestamp node t_1 (i.e., <T t="i">) and attach children(y) to t_1;
attach t_1 as child node of x;

else
let $XY = \{(x', y') | x' \in \text{children}(x), y' \in \text{children}(y), \text{label}(x') = \text{label}(y')\}$,
let X' denotes the rest of the nodes in children(x) that do not occur in XY
and Y' denotes the rest
of the nodes in children(y) that do not occur in XY.
for every pair $(x', y') \in XY$
(a) Nested-merge (x', y', T)
for every $x' \in X'$
(b) **If** time(x') does not exist **then** time(x') := $T - \{i\}$;
for every $y' \in Y'$
(c) time(y') := i and attach y' as a child node of x;

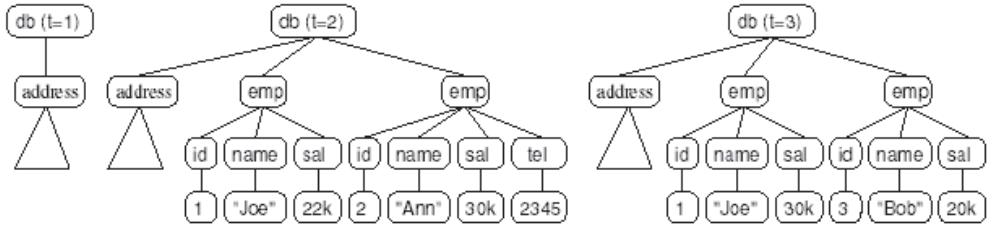

Figure 7.1: A sequence of versions of a company database.

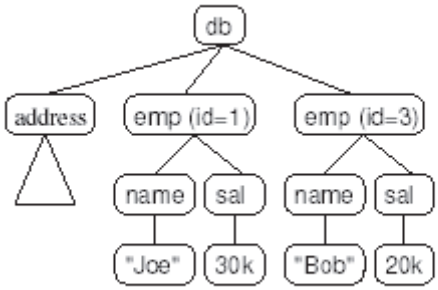

Figure 7.2: Version 3 annotated with key values.

A simple example illustrates how the algorithm works.

Example 6: Figure 7.3 shows an example of nested merge when version 3 in Figure 7.1 is merged into the archive containing versions 1 and 2. The algorithm first determines the current set of timestamps. It is i added to time(x) if time(x) exists. Otherwise,

timestamps are inherited from its parent. It is a property of the algorithm that label(x) and label(y) are the same whenever Nested-Merge (x, y, T) is invoked. The algorithm then proceeds by checking if y is a frontier node. As an example, `sal` is a frontier node. In Figure 7.3, when version 3 is merged, the value of `sal` of `Joe` differs from that in the archive. Hence timestamps are used to enclose the salary values at the respective times in the new archive.

The children nodes of any frontier node are all timestamps nodes or none of them is a timestamp node. In Figure 7.3, `sal` of Joe is an example of a frontier node whose children are all timestamps nodes, `tel` of `Ann` is a frontier node none of whose children are timestamp nodes. If y is not a frontier node, we partition nodes in children(x) and children(y) into three sets:

- XY contains pairs of nodes from children(x) and children(y) respectively with the same label and key value. Nested merge is recursively called on pairs of nodes in XY, inheriting the current timestamp T.
- X' consists of nodes in children(x) where does not exist any node in children(y) with an equal key values. To ensure that nodes of X' no longer exist at time i, timestamp T excluding i is annotated on nodes of X' if they do not already contain timestamps that terminate earlier than i.
- Y' consists of nodes in children(y) where does not exist any node in children(x) with an equal key value. Subtrees rooted at nodes of Y' are attached as a subtrees of x and they are annotated with a timestamp i since they only begin to exist at time i.

Observe that in the resulting archive

- nodes appearing in many versions are stored only once in the archive.
- if the node occurs in version 3, then the timestamp of the corresponding node in the archive contains 3. Note that the node `emp` (`id =3, t=[3]`) contains only the timestamp `t=3` since it does not have a corresponding node in the old archive.
- time intervals is used to describe the sequence of versions for which a node exists. For example, the time interval `[1-3]` denotes the set $\{1, 2, 3\}$.
- if a node does not have timestamp, it is assumed to inherit the timestamp of its parent. For example, the name node under the `emp` (`id =1, t=[2-3]`)inherits the timestamp `t=[2-3]`.
- the timestamp of a node is always a superset of timestamps of any descendant node.

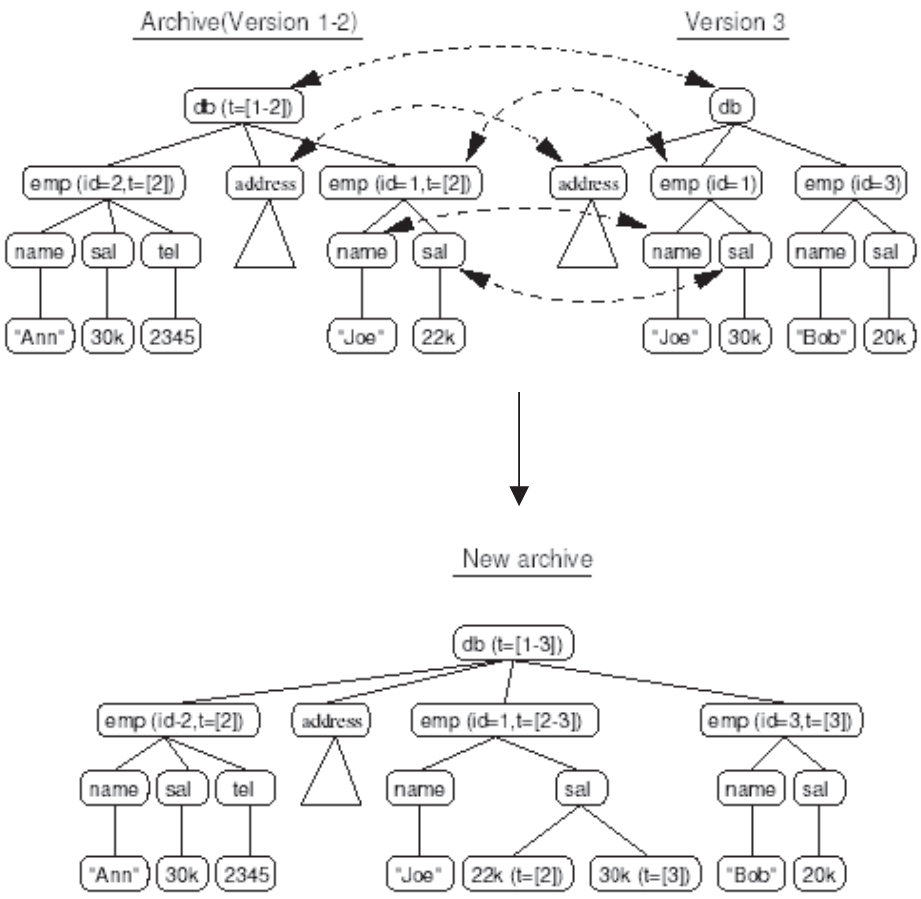

Figure 7.3: Merging version 3 into the archive containing versions 1 and 2.

The XML representation of the new archive in Figure 7.3 is as follows.

```
<T t= 1-3 >
      <db>
           <address> ...</address>
           <T t= 2-3 >
                <emp><id>1</id>
                <name>Joe</name>
                <sal><T t= 2 >22k</T><T t= 3 >30k</T></sal></emp>
           </T>
           <T t= 2 >
                <emp><id>2</id>
                <name>Ann</name>
                <sal>20k</sal><tel>2345 </tel> </emp></T>
           <T t= 3 >
                <emp><id>3</id>
                <name>Bob</name>
                <sal>20k</sal> </emp></T>
      </db>
</T>
```

7.3 Querying

Since the archive is in XML, the existing XML query languages such as XQuery can be used to query such documents. The authors of [3] did not discuss the issue of temporal queries.

Chapter 8

A Multidimensional XML Model (MXML)

The proposed approach [5] can represent multiple versioning not only with respect to time but also to other context parameters such as language, degree of detail, etc. The model is capable to represent changes not only in an XML document but in the corresponding XML schema as well. However, how to support queries is not discussed in [5].

8.1 Data Modelling

In a multidimensional XML document, dimensions may be applied to elements and attributes. A multidimensional element/attribute is an element/attribute whose contents depend on one or more dimensions. The notion of *world* is fundamental in MXML. A *world* represents an environment under which data in a multidimensional document obtain a meaning. A *world* is determined by assigning values to a set of dimensions.

Definition 7: Let S be a set of dimension names and for each $d \in S$, let D_d, $D_d \neq \emptyset$, be the domain of d. A *world* W is a set of pairs (d, u), where $d \in S$ and $u \in D_d$ such that for every dimension name in S there is exactly one element in W.

Example 7: Consider the *world* $w = \{$(*time*, 2005-12-14), (*customer_type*, student), (*edition*, English). The dimensions names are *time, customer_type*, and *edition*. The assigned values of these dimensions names are 2005-12-14, student, and English respectively.

The syntax of XML is extended in order to incorporate dimensions. A multidimensional element has the form:

```
<@element-name attribute-specification>
      [context-specifier-1]
      <element-name attribute-specification-1>
          element-content-1
      </element-name>
[/]
. . .

      [context-specifier-N]
      <element-name attribute-specification-N>
          element-content-N
      </element-name>
[/]
</@element-name>
```

To declare a multidimensional attribute the following syntax is used:

```
attribute-name =   [context-specifier-1] attribute-value-1[/]
. . .
                   [context-specifier-n] attribute-value-n[/]
```

The multidimensional element is denoted by preceding the element's name with the special symbol "@", and encloses one or more context elements. All context elements of a multidimensional element have the same name which is the name of the multidimensional element.

Context specifiers qualify the facets of multidimensional elements and attributes, called context elements/attributes, stating the sets of worlds under which each facet may hold. The context specifiers of a multidimensional element/attributes are considered to be mutually exclusive, in other words they must specify disjoint sets of worlds.

The time period during which a context element/attribute is the holding facet of the corresponding element/attribute is denoted by qualifying the context element/attribute with context specifier of the form `[time in{`$\mathtt{t}_1$ `...`$\mathtt{t}_2$`}]`, It is assumed that:

i. A dimension named `time` is used to represent time. The time domain `T` of time is linear and discrete; $\mathtt{t}_1$ and $\mathtt{t}_2$ represent the start time and end time respectively.
ii. A reserved value`start`, such that `start` $<$ `t` for every `t` $\in$ `T`, representing the beginning of time.
iii. A reserved value `now`, such that `t` $<$ `now` for every `t` $\in$ `T`, representing current time.

Figure 8.1 shows an instance of MXML document. In this example, the element `book` has six subelements:

- The `isbn` and `publisher` are multidimensional elements and depend on the dimension `edition`. The multidimensional element `@isbn` has two context elements having the same name `isbn` (without the special symbol "`@`"). `[edition = greek]` and `[edition = English]` are the context specifiers of `@isbn`.

```
<book>
   <@isbn>
        [edition = greek] <isbn>0-13-110370-9</isbn> [/]
        [edition = English] <isbn>0-13-110362-8</isbn> [/]
   </@isbn>
   <title>The C programming language</title>
   <authors>
        <author>Brian w. Kernighan</author>
        <author>Dennis M. Ritchie</author>
   </authors>
   <@publisher>
        [edition = English]<publisher>prentice Hall</publisher>
        [/]
        [edition = greek]<publisher>Klidarithmos</publisher>
        [/]
   </@publisher>
   <@translator>
        [edition =greek]<translator>Thomas Moraitis</translator>
        [/]
   </@translator>
   <price currency=[edition=greek,time in{start..2001-12-31}]
                        GRD[/]
                   [edition = greek,time in{2002-01-01..now}]
                        EURO[/]
                   [edition = English]USD[/]>
    <@value>
        [edition = greek,time in{start..2001-12-31}]
                   <value>13.000</value>[/]
        [edition = greek,time in{2002-01-01..now}]
                   <value>40</value>[/]
        [edition = English]
                   <value>80</value>[/]
    </@value>
    <@discount>
        [edition = greek, customer_type=student]
                   <discount>20</discount>[/]
        [edition = greek, customer_type = library]
                   <discount>10</discount>[/]
    <@discount>
   </price>
</book>
```

Figure 8.1: Multidimensional Information about a book encoded in MXML.

- The elements `title` and `authors` are conventional elements (remain the same under every possible world).
- The element `price` is a conventional element containing a multidimensional attribute (the attribute `currency`) and two multidimensional elements (`value` and `discount`). The value of the attribute `currency` depends on the dimensions edition and time (as to buy the English edition we have to pay in `USD`, while to buy the `Greek` edition we should pay in `GRD` before `2002-01-01` and in `EURO` after that date due to the change of the currency in EU countries).
- The element `value` depends on the dimensions `edition` and `time`, while the element `discount` depends on dimensions `edition` and `customer_type`.

As we referred before, the context specifiers of a multidimensional element/attribute must be mutually exclusive. This property makes it possible, given a specific world, to reduce an MXML document to an XML document holding under that world. Informally the reduction of an MXML document D to an XML document D_w holding under the world w proceeds as follows:

- Each multidimensional element E is replaced by its context element E_w (the value of E_w represents the value of E under the world W). If there is no such context element, then E along with its subelements is removed entirely.
- A multidimensional attribute A is transformed into a conventional attribute A_w whose name is the same as A and whose value is the holding one under W. If no such value exists then the attribute is removed entirely.

Example 8: For the world $w = \{$(*time*, 2002-03-03), (*customer_type*, student), (*edition*, greek)$\}$, the MXML document in Figure 8.1 is reduced to the conventional XML document that follows:

```
<book>
      <isbn>0-13-110370-9</isbn>
      <title>The C programming language</title>
      <authors>
          <author>Brian W. Kernighan</author>
          <author>Dennis M. Ritchie</author>
      </authors>
      <publisher>Klidarithmos</publisher>
      <translator>Thomas Moraitis</translator>
      <price currency = EURO>
          <value>40</value>
          <discount>20</discount>
      </price>
</book>
```

8.2 Data Modification

Three primitive change operations (update, delete, insert) on XML documents can be represented in MXML for both elements and attributes.

Basic change operations on elements

The deletion of the element r at time t is represented by:

i. changing the end time point of the most recent facet of the element (the facet for which the end point of the value of d is now) from now to `t-1` if r is already multidimensional element, or
ii. a multidimensional element with a single facet holding during the interval {`start..t-1`} as shown in Figure 8.2.

Figure 8.3 shows applying the operation update on the element `r` at time `t` (note that a dimension named `d` is used to represent time).

Figure 8.2: Applying the operation delete on the element `r` at time `t`.

Basic change operations on attributes

The basic change operations on attributes are similar to those on elements. Consider the element v1 , and suppose we want to

i. delete the attribute a_1 at time point `t`. Then we get the following MXML element:
<p a1 =[d in{start..t-1}]‘‘9’’ [/]> v1
ii. add at time point `t`, a new attribute whose name is a_2 and whose value is 3. Then we get the following MXML element:
<p a1 = ‘‘9’’ a2 = [d in {t..now}] ‘‘3’’ [/]> v1
iii. update the value of the attribute a_1 to the new value 8 at time point `t`, then we obtain the following MXML element:
<p a1 =[d in{start..t-1}]‘‘9’’[/][d in {t..now}]‘‘8’’[/]> v1

Basic change operations on an XML schema

Changes in an XML document often require corresponding changes in the document's schema. The history of the schema of an XML document can be represented easily, for instance deleting an element, or adding an attribute to an element at a specific time point.

Example 9: The following XML schema description adds the fixed value attribute r_1 to the element `r` during the interval {`t..now`}.

```
<xs:element name="r" type="xs:string" r1=[d in {t..now}]"0" [/]/>
```

Example 10: Consider the XML document in the left side of Figure 8.2. The schema for this document may be encoded in XML schema as follows:

```
<xs:element name="p">
      <xs:complexType>
          <xs:sequence>
              <xs:element name="q" type="xs:string"/>
              <xs:element name="r" type="xs:string"/>
              <xs:element name="s" type="xs:string"/>
              </xs:sequence>
      </xs:complexType>
</xs:element>
```

After deleting the element <r>v_2 </r> (as described in Figure 8.2), it is necessary to modify the document's schema if we want the XML document resulting by applying the deletion to become valid. This change can be represented by turning the element sequence of the above XML schema into a multidimensional element with two facets:

```
<xs:element name="p">
      <xs:complexType>
          <@xs:sequence>
              [d in {start.. t-1}]
                  <xs:sequence>
                      <xs:element name="q" type="xs:string"/>
                      <xs:element name="r" type="xs:string"/>
                       <xs:element name="s" type="xs:string"/>
                  </xs:sequence>
              [/]
              [d in { t..now}]
                  <xs:sequence>
                      <xs:element name="q" type="xs:string"/>
                      <xs:element name="s" type="xs:string"/>
                  </xs:sequence>
              [/]
          </@xs:sequence>
      </xs:complexType>
</xs:element>
```

```
<p>
      <q> v1 </q>
      <r> v2 </r>
      <s> v3 </s>
</p>
```

```
<p>
      <q> v1 </q>
      <@r>
            [d in {start..t-1}]
            <r> v2 </r> [/]
            [d in {t..now}]
            <r> v4 </r> [/]
      </@r>
      <s> v3 </s>
</p>
```

Figure 8.3: Applying the operation delete on the element `r` at time `t`.

Chapter 9

Summary of XML-based Temporal Data Models

So far, we have introduced some works which have made important contributions in providing expressive and efficient means to model, store, and query XML-based temporal data models. All the mentioned models are summarized in Figure 9.1. All the models are capable to represent changes in an XML document by supporting temporal elements, and incorporating time dimensions. Two time dimensions are usually considered: valid time and transaction time. In [7] a publication time and efficiency time in the context of legal documents are proposed. Time dimensions may be applied to elements and attributes. In [5] and [9] the temporal attributes are supported. In our point of view, supporting temporal attributes adds an advantage to the model.

Temporal information is supported in XML much better than relational tables. This property is attributed to the hierarchical structure of XML which is compatible perfectly with the structure of temporal data. Only in [5] the syntax of XML is extended in order to incorporate not only time dimensions but also other dimensions such as language, degree of detail, etc. So the approach in [5] is more general than other approaches as it allow the treatment of multiple dimensions in a uniform manner.

In our point of view, the model's power depends also on supporting powerful temporal queries. In [9] and [10] powerful temporal queries expressed in XQuery without requiring the introduction of new constructs in the language are supported. In [11] a valid time support is added to XPath. This support results in an extended data model and query language. The other models in [5], [7], and [3] did not discuss the issue of temporal queries. A significant advantage will be added to the model if it is not only representing the history of an XML document but also the history of its corresponding XML schema or DTD as well. In [5], [7], and [9] the temporal XML schema/DTD is supported by extending the existing XML schema/DTD.

We conclude that XML provides a flexible mechanism to represent complex temporal data. XML can be even an option for implementations of temporal databases (or multi-dimensional databases) on a top of a temporal XML database. Our work shows that there are a lot of important topics for forthcoming research. Many research issues remain open at the implementation level, including the use of nested

XML-based Temporal Models	**Considered features**					
	Time Dimension	**Temporal elements**	**Temporal attributes**	**Querying**	**XML schema /DTD**	**XML syntax**
XBIT XML-based bitemporal data model	valid/ transaction time	supported	not supported	XQuery	-	not extended
XML-based model for versioned document	valid/ transaction time	supported	supported	XQuery	are extended	not extended
XML-based model for versioned normative texts	valid/ transaction/ publication/ efficacy time	supported	not supported	-	XML-schema is extended	not extended
Valid time XPath data model	valid time	supported	not supported	XPath is extended	-	not extended
XML-based model for archiving data	valid time	supported	not supported	-	-	not extended
MXML Multidimensional XML model	valid/ transaction time	supported	supported	-	XML-schema is extended	extended

Figure 9.1: Summary of XML-based temporal data models.

relations on the top of an object-relational DBMS, reflecting temporal features into an associated XML query language etc.

Bibliography

[1] Boag, S., Chamberlin, D., Fernández, M. F., Florescu, D., Robie, J. Siméon, J.: XQuery 1.0: An XML Query Language, W3C Working Draft, 04 April 2005. Dostupné na: `http://www.w3.org/TR/xquery/`.

[2] Bourret, R.: XML and Databases, 2004. Available: `http://www.rpbourret.com/xml/XMLAndDatabases.htm`.

[3] Buneman, P., Khanna, S., Tajima, K., and Tan, W.: Archiving scientific data. In proc. of ACM SIGMOD Int. conference, pp. 1-12, 2002.

[4] Dyreson, C.E. Observing Transaction-Time Semantics with TTXPath. In proc. of the 2nd Int. conference on Web Information System Engineering (WISE), pp. 193-202, 2001.

[5] Gergatsoulis, M. and Stavrakas, Y.: Representing Changes in XML Documents using Dimensions. In proc. of 1st Int. XML Database symposium (Xsym), pp. 208-221, 2003.

[6] Grandi, F. and Mandreoli, T.: The valid web: An XML/XSL Infrastructure for Temporal Management of Web Documents. In Proc. of Int. conference on Advances in Information Systems (ADVIS), pp. 294-303, 2000.

[7] Grandi, F., Mandreoli, T., and Bergonzini: A Temporal Data model for the management of normative texts. In proc. SEBD2003-Natl' conference on Advanced Database Systems, pp. 169-178, 2003.

[8] Tansel, A., Clifford, J., Gadia, S., Jajodia, S., Segev, A., Snodgrass, R.T.: Temporal Databases: Theory, Design and Implementation, Benjamin/Cummings Publishing Company, California. pp. 496-507, 1993.

[9] Wang, F. and Zaniolo, C.: Temporal Queries in XML Document Archives and Web Warehouses. In proc. of 10th Int. Symposium on Temporal Representation and Reasoning (TIME-ICTL 2003), pp. 47-55, 2003.

[10] Wang, F. and Zaniolo, C.: XBIT: An XML-based Bitemporal Data Model . In proc. of 23rd Int. conference on Conceptual Modeling (ER2004), 2004.

[11] Zhang, S. and Dyreson, C.: Adding Valid Time to XPath. In proc. of Database and Network Information Systems (DNIS), pp. 29-42, 2002.

[12] W3C: Extensible Markup Language (XML) 1.1. W3C Recommendation 04 February 2004, edited in place 15 April 2004. Available: `http://www.w3.org/TR/xml11/`.