

Temporal XML

Presentation and discussion of the approach by F. Rizzolo and A. Vaisman

Paper for the seminar in Database Systems

Universität Zürich
Institut für Informatik
30. Mai 2010

written by
Kristin Kruse
04-720-983

1 Introduction

This following work is a discussion of the article *Temporal XML: modelling, indexing and query processing* by F. Rizzolo and A. Vaisman, which has been published 2008 in the International Journal on Very Large Data Bases (VLDBJ), 17(5). This article is specifying an earlier publication made by A. Mendelzon, F. Rizzolo and A. Vaismann calls *Indexing Temporal XML Documents* which has been published 2004 at the 30th International Conference on Very Large Data Bases (VLDB) and in which TempIndex has been introduced the first time. The authors are researching in the field of indexing schemes for temporal and non-temporal XML for some time.

The meaning of this following text is to present the approach made in the article *Temporal XML: modelling, indexing and query processing* but by restricting to the main idea and its consequences. This involves the enlargement from XML to temporal XML considering a document and its impact to indexing and query processes. The new indexing scheme TempIndex is presented by introducing its foundations and there is a larger example to enlighten the query process. Skipped parts of the discussed article are mentioned in the section 7 The approach of F. Rizzolo and A. Vaisman. Therefore a bigger part at the beginning is dedicated to a general introduction of XML.

2 XML document

2.1 General

The shortcut “XML” stands for eXtensible Markup Language. A Markup Language is used to structure documents, above all text, by dividing the input into a sequence of describing parts e.g. title, subtitle, etc. A part can than recursively be divided in additional parts e.g. a paragraph contents several sentences.

XML is used in this meaning but not only on text, but on all contents that diverge in substance so that it can't be represented in a standard database.

2.2 Data model

The XML brings along *elements*, *attributes* and *values* which are represented by nodes in the data model.

- An *element* includes an *element tag* that allocates it as an instance of a specific element entity. An element can contain other elements and attributes.
- An *attribute* also includes an *attribute tag* that allocates it as an instance of a specific attribute entity. There can be an additional annotation of ID or REF, but there purpose is not discussed in this text. An attribute only contains a final value.

- A *value* always belongs to exactly one attribute and stores the final values in a requested format.

The relationships in the data model are expressed by different edges:

- A *containment edge* $e_c(n_i, n_j)$ is an edge in the tree. It basically expresses that the node at the top of the edge n_i contains the node at the bottom of the edge n_j .
- A *reference edge* $e_r(n_i, n_j)$ is a connection between an attribute node n_i of type REF and its corresponding element node n_j . This kind of connection is used neither in the non-temporal nor in the temporal data model. It is only brought to bear at the implementation of temporal XML (Sect. 3.3).

A XML document is the final product and describes a certain content by a sequence of recursively nested instances. Fig. 1 shows a extraction of a XML document with the element entities *lecture*, *tutorage*, *person* and the attribute entity *name*.

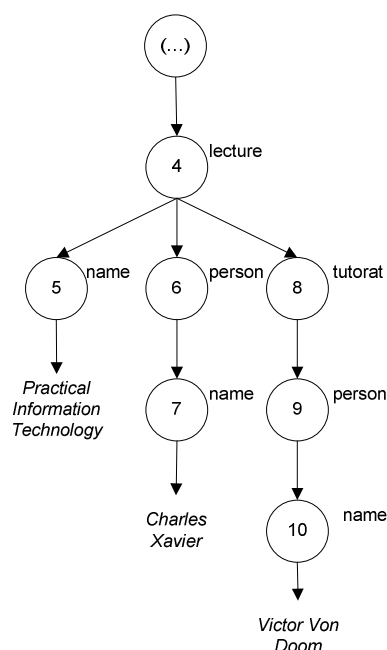


Fig. 1. An extraction of a XML document: one lecture element with its subelements and attributes.

Every XML document can be illustrated as a tree. An internal node is representative for one element, its subtree is its specification and the leaf nodes represent the attributes with the final values.

2.3 Implementation

An institute of a university wishes to present their teaching program for each semester on their homepage. The lectures displayed are periodically held in a particular semesters. Each lecture is specified by several details, some of them are obligatory, others are optional. Some are composed out of multiple, others out of single information. Along the lecture and its details there should be provided search possibilities.

In Order to structure the teaching programs for proper presentation and useful queries, there is used the eXtensible Markup Language. The following implementation shows a possible allocation out of the introduced example above by using the element teaching for the document, partOfYear for the semester division, lecture, person and tutorage. There is the attribute name, when strings need to be stored.

```

<?xml version="1.0"?>
<teaching>
  <partOfYear ID="1">
    <name >Spring</name>
    <lecture ID="4">
      <name>IT: IIa</name>
      <person ID="6">
        <name>Charles Xavier</name>
      </person>
      <tutorage>
        <person ID="9">
          <name>Victor Von Doom</name>
        </person>
      </tutorage>
    </lecture>
    <lecture ID="11">
      <name>Practical Information Technology</name>
      <person ID="13">
        <name>Henry Jones Jr.</name>
      </person>
    </lecture>
    <lecture ID="20">
      <name>Informatik IIb</name>
      <person ID="25">
        <name>Abraham Van Helsing</name>
      </person>
    </lecture>
  </partOfYear>
</teaching>
  
```

```
<tutorage>
  <person ID="23">
    <name>Who</name>
  </person>
</tutorage>
</lecture>
</partOfYear>
<partOfYear ID="2">
  <name >Autumn</name>
  <lecture ID="16">
    <name>Informatik I</name>
    <person ID="18">
      <name>Hubert J. Farnsworth</name>
    </person>
  </lecture>
</partOfYear>
</teaching>
```

Fig. 2. A possible XML document describing the introduced example

BUT: when a semester is out of date, its program should not be deleted but stay available along the next active and the already passed programs for archive functions.

To implement this with non-temporal XML would mean to create a XML document for each semester. This causes every semester an increase to the size at the homepage and at the same time stores a lot of redundant information, because periodically held lectures are implemented multiple times this way. It also effects the searching which is only possible inside one semester and there is no possibility to track historical changes along a timeline e.g. one same lecture is held over time, but had changes in their teaching person.

3 Temporal XML document

3.1 General

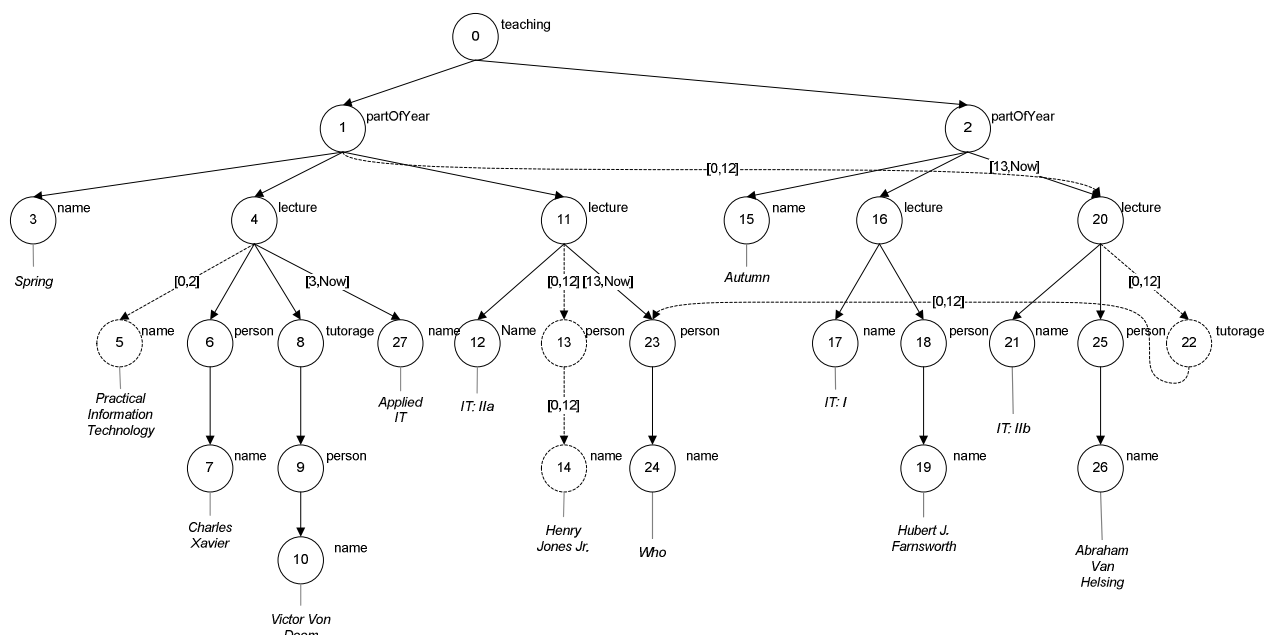
The limits of the non-temporal XML are about to be reached as soon as there is a historical level that has to be involved in the XML document. Therefore one big research branch in XML topics goes to the implementation of a temporal extension to non-temporal XML.

F. Rizzolo and A. Vaisman's proposition is one such temporal XML extensions.

Their approach is it, to expend the elements and attributes in the non-temporal XML by a *temporal interval* in which the temporal validity of an element or attribute as subpart of superior element is shown. This interval is a ordered in pair and will be formally written as $[a, b]$ where a is the starting point and b the ending point in a timeline. This is calculated in transaction time, so that still valid connections at the current timestamp are denoted with the expression *Now* as ending point.

Including this temporal interval it is possible to represent multiple states of a XML document over time in one single document. If changes occur, they don't override the old information but are added to the document.

Fig. 3. The temporal data model representing the XML document from Fig. 2 but with some additional temporal changes



3.2 Data model

In the data model, the temporal interval would be attached to the containment edges $e_c(n_i, n_j, T)$ by extending it with a *temporal label* $T = [T_{ec}.FROM, T_{ec}.TO]$ which expresses the start and if terminated the end of the time in which n_j was contained by n_i .

In the presented article the authors are making the restriction of dealing only with one (plain) interval per edge instead of *interval elements* (set of intervals) for the sake of simplification and implementation efficiency. With this as precondition, they developed the following definition for temporal XML:

“Definition 3 (temporal XML document) *A Temporal XML Document* is a document graph augmented with temporal labels that satisfies the following conditions:

1. The union of the temporal labels of the containment edges outgoing from a node is contained in the lifespan¹ of the node.
2. The temporal labels of the containment edges incoming to a node are consecutive.
3. For any time instant t , the sub-graph composed by all containment edges e_c such that $t \in T_{ec}$ is a tree with root r . We call this subgraph a *snapshot* of the document at time t , denoted $D(t)$:
4. For any containment edge $e_c(n_i; n_j; T_{ec})$; if n_j is a node of type ID, the time label of e_c is the same as the lifespan of n_i ; moreover, if there are two elements in the document with the same value for an ID attribute, both elements are the same. In other words, the ID of a node remains constant for all the snapshots of the document.
5. For any containment edge $e_c(n_i; n_j; T_{ec})$; if n_j is an attribute of type REF, such that there exists a reference edge $e_r(n_j; n_k; T_{er})$; then $T_{ec} = T_{er}$ holds.
6. Let $e_r(n_i; n_j; T_{er})$ be a reference edge. Then, $T_{er} \subseteq \text{lifespan}(n_i)$ holds.”

(Rizzolo, F.; Vaisman, A. (2008) p.1184)

Point 1 declares that a node is only valid as long as it belongs to a valid supernode.

Point 2 includes the previously mentioned precondition: if the interval of the incoming edges to a single node would leaf gaps between them, the lifespan of a node, like the interval of the edges leading to its subnodes couldn't be stored in a plain interval. From this it follows that a node can never “take a break” of validity and become valid again.

Point 3 includes the same precondition: if the interval of the incoming edges to a single node would intersect, there exist some timestamps in which a node has more than one supernode and the snapshot would not be a tree. From this it follows that a node can never have multiple supernodes at the same time.

Point 4, 5 and 6 declare the use of system attributes and reference edges, which are not going to be discussed here.

Fig. 3 shows an example of a temporal XML document. For better visual understanding, all the currently active containment edges (= valid until *Now*) are shown as black lines and passed containment edges are dashed lines.

Unlike a non-temporal XML document, a temporal XML document can no longer be illustrated by a tree since there are now nodes that can be associated to different supernodes at different times e.g. the person with the ID 23 previously held a tutorage and is now lecturer of the lecture *IT:IIa*. A temporal XML document presents himself as a top-bottom directed graph.

But point 3 of the definition declares that a snapshot starting at the document's root for one specific timestamp is a tree since every node has only one valid supernode at one time.

This basically means that a temporal XML document is the set of multiple non-temporal XML documents – one for every different timestamp.

¹ Lifespan of a node = the time in which the node is valid inside the document.

3.3 Implementation

There are many different ways of implementing a XML document. The here presented article includes four different versions in order to implement temporal XML. One of it, the non-replicated implementation, is used to enlighten their example. This is used throughout this paper as well.

In all versions of implementation, the temporal label is going to be stored as a system attribute at the proper element or attribute.

For simplicity the temporal interval has so far been denoted in relative timestamps. In real life application the starting point and the ending point would contain time and/or date expressions. Rizzolo and Vaisman defined for their applications an own namespace *Time*, which is applied in the following example for illustration.

```
<?xml version="1.0"?>
<teaching xmlns:
Time="http://www.cs.Toronto.edu/db/time">
  <parTofYear ID="1" Time:FROM="1990-01-01"
Time:TO="Now">
    <name Time:FROM="1990-01-01"
Time:TO="Now" >Spring</name>
    <lecture ID="4" Time:FROM="1990-01-01"
Time:TO="Now">
      <name Time:FROM="1990-01-01"
Time:TO="1992-12-31">Practical Informa-
tion Technology</name>
      <name Time:FROM="1993-01-01"
Time:TO="Now">Applied IT</name>
      <person ID="6" Time:FROM="1990-01-01"
Time:TO="Now">
        <name Time:FROM="1990-01-01"
Time:TO="Now">Charles Xavier</name>
      </person>
      <tutorage Time:FROM="1990-01-01"
Time:TO="Now">
        <person ID="9" Time:FROM="1990-01-01"
Time:TO="Now">
          <name Time:FROM="1990-01-01"
Time:TO="Now">Victor Von Doom</name>
        </person>
      </tutorage>
    </lecture>
    <lecture ID="11" Time:FROM="1990-01-01"
Time:TO="Now">
      <name Time:FROM="1990-01-01"
Time:TO="Now">IT: IIa</name>
      <person ID="" Time:FROM="1990-01-01"
Time:TO="2002-12-31">
```

```
      <name Time:FROM="1990-01-01"
Time:TO="2002-12-31">Henry Jones
      Jr.</name>
    </person>
    <person Time:FROM="2003-01-01"
Time:TO="Now" IN = "23"/>
  </lecture>
  <lecture ID="20" Time:FROM="1990-01-01"
Time:TO="2002-12-31">
    <name Time:FROM="1990-01-01"
Time:TO="Now">IT: IIb</name>
    <person ID="" Time:FROM="1990-01-01"
Time:TO="Now">
      <name Time:FROM="1990-01-01"
Time:TO="Now">Abraham Van Helsing
      </name>
    </person>
    <tutorage Time:FROM="1990-01-01"
Time:TO="2002-12-31">
      <person ID="23" Time:FROM="1990-01-
01" Time:TO="2002-12-31">
        <name Time:FROM="1990-01-01"
Time:TO="Now">Who</name>
      </person>
    </tutorage>
  </lecture>
  <parTofYear>
  <parTofYear ID="2" Time:FROM="1990-01-01"
Time:TO="Now">
    <name Time:FROM="1990-01-01"
Time:TO="Now" >Autumn</name>
    <lecture ID="16" Time:FROM="1990-01-01"
Time:TO="Now">
      <name Time:FROM="1990-01-01"
Time:TO="Now">IT: I</name>
      <person ID="18" Time:FROM="1990-01-01"
Time:TO="Now">
        <name Time:FROM="1990-01-01"
Time:TO="Now">Hubert J. Farns-
worth</name>
      </person>
    </lecture>
    <lecture Time:FROM="2003-01-01"
Time:TO="Now" IN = "20"/>
  </parTofYear>
</teaching>
```

Fig. 4. The *teaching* example implemented temporal XML document (non-replicated representation)

As mentioned before, non-replicated representation is used here. This way of implementation has only one entry for one node in the data model. But the temporal XML data model allows associations to multiple supernodes in different temporal intervals, and they all need to be displayed in the document. So only the oldest appearance is being implemented. The later ones are expressed by implementing a reference attribute that references to the *oldest appearance* (See in Fig. 4: the person who is taking over the lecture with ID 11 is

already stored as tutor with ID 23 of another lecture. The person element under 11, is only referring to 23.)

4 Indexing

4.1 General

In order to avoid traversing a whole XML document in query processes it is useful to work with indexing schemes. Its entries, referring to the actual elements or attributes in the document, are sorted or filtered and provide an easier and faster way of access. These schemes are generated once out of a XML document and are stored separately. This implies that they need to be updated when changes in the document occur.

4.2 Continuous path

The index entries take advantage of the directed graph-properties of the temporal XML document, by calculating with paths from the root leading to every node in the graph.

In non-temporal XML, where every document equals a tree, every node could be addressed by exactly one path. Considering a temporal XML document as a set of non-temporal XML documents – one for every timestamp (see chapter 3.2) there is also exactly one path leading every node, at least at a specific timestamp.

A *continuous path* $p(n, I)$ is the path from the root to a specific node n including the temporal interval of the path I . It means that this path addresses this node n during the time of the interval I . I is the validity of the path and can be computed as the intersection of the temporal intervals of all visited edges.

The difference to normal paths is that a continuous path includes not only the information *where* the path leads to, but also *when*. A node with multiple parents has multiple continuous paths leading to the node. But every one of these continuous paths is valid at a

different temporal interval since the parents are never valid at the same time. So by combining a path with its temporal validity we can deal with a set of continuous paths, where every continuous path leads to a unique result, like a normal path in a non-temporal XML document.

4.3 Temporal interval encoding

Before entering the indexing scheme, there should be introduced the *temporal interval encoding* which takes advantage of the ancestor-descendent relations between nodes and its subtrees. The basic idea is to encode every continuous path to a node in relation to the nodes ancestor (= supernode that is valid in this specific continuous path). There is an assignment of ascending numeric values whilst traversing the document graph in pre-order. So every subnode has a bigger value than its parent (See Fig. 5). There are as many encoding values as there are continuous paths and not as there are nodes. Like that nodes with several parents are getting visited several times. By using the encoding values instead of the ID number the information about the ancestor-descendent relationship is included in every continuous path. In a query this can be used very effectively. To allow later insertions and updates concerning new continuous paths there can be left gaps between the values².

² The logic of the gaps is in the original article not explained. For this paper the author has predefined the size of subtrees for every element.

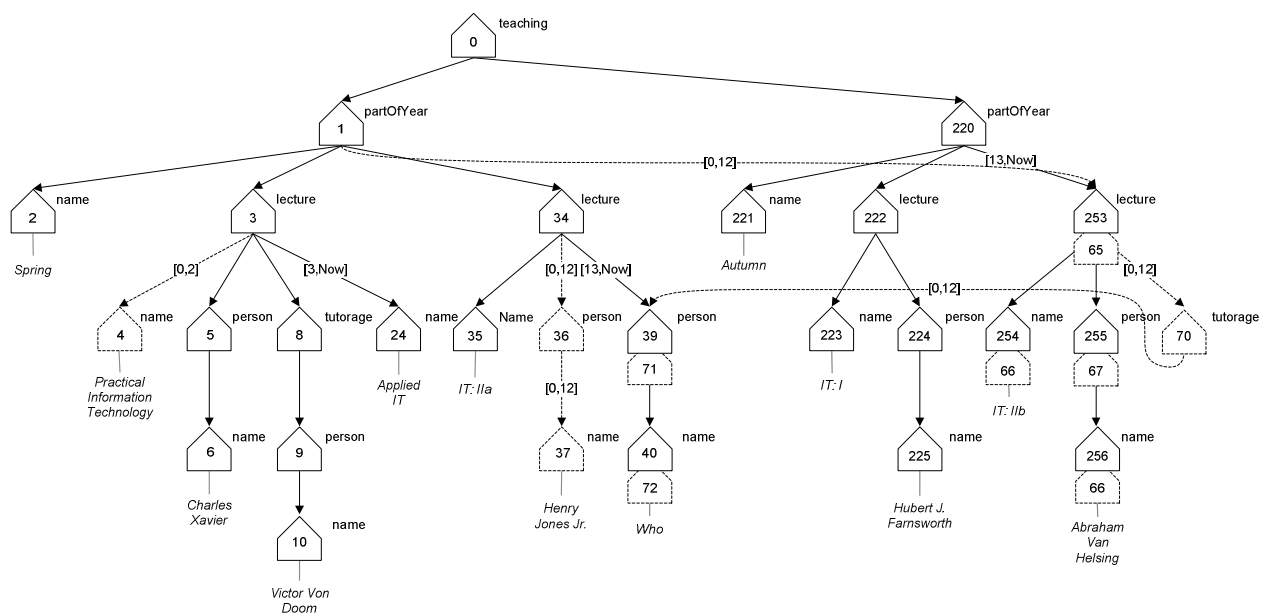


Fig. 5. The temporal data model of the same temporal XML document as in Fig. 3 but this time with temporal interval encodings of the continuous paths.

4.4 TempIndex

TempIndex is the indexing scheme for temporal XML developed by A. Mendelzon, F. Rizzolo and A. Vaismann. It stores continuous paths with temporal interval encoding in a combination of several data structures (different filters and sorting list) to provide different access possibilities during a query process. The used data structures are being introduced in the following section.

Summaries

The data structure *summary* has been developed for indexing of non-temporal XML documents and deals with normal path. For the use with temporal XML documents it has been extended to deal with continuous paths. The extended version is called *temporal summary*.

Temporal summary of a document D , $SD = (TSum; tmap; edge; Label; \lambda)$

TSum: a set of equivalence classes

Tmap: relation that assigns every continuous path to exactly one equivalence class, noted as a tuple [TSum, n, l]

Edge: stores the horizontal relation between the equivalence classes

Label: list of the labels of the equivalence classes

λ : function that assigns the equivalence classes with labels from the list *label*.

The idea is to summarize the horizontal structure of a document by a certain condition, so that there is a simpler structure left that can be displayed in the so-called *summary graph* (Fig. 6) which is stored indirectly in TSum, Edge and Label. Every node in the summary graph is considered as an equivalence class and every continuous path can be assigned to exactly one of these classes.

In TemplIndex the authors have defined two different temporal summaries:

The first one is *LCP summary*³ and summarizes continuous paths from the root to a node by same label (Fig. 6, left).

The second temporal summary is the *interval summary* (Fig. 6, right). In this case the continuous paths are clustered into equivalences classes of a certain path depth. And because a

³ The shortcuts in names aren't explained in the original article. LCP will probably mean: labelled continuous path.

continuous path is a path in a tree, this corresponds to the path length.

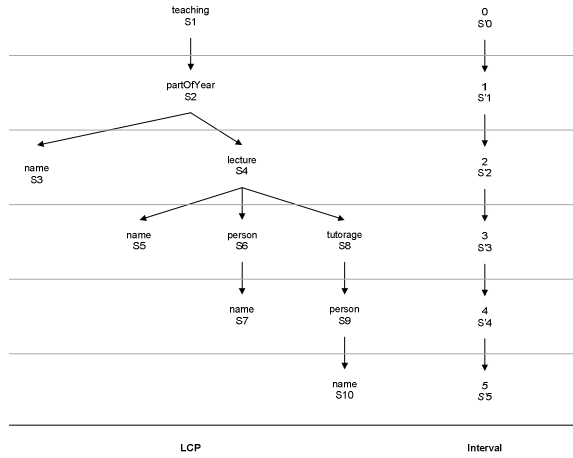


Fig. 6. Summary graphs of the LCP summary and the Interval summary from the temporal XML document *teaching*

Tables

TempIndex extends the two mentioned TSummaries by additional data structures, namely tables. There is table for every equivalence class of the summaries. But the tables not just store the output of the summary but allow it to store additional information or present additional way of access. In these tables the temporal interval encodings are of great importance.

First, there is the δ_k table for every summary node in the interval summary, means for every temporal depth k . These tables store all the continuous paths of the corresponding depth but summarized a second time according to their interval of validity. The tuples t inside every table consist of the attributes *from*, *to* and a list called *valid*. Every tuple defines a unique interval $[from, to]$ and stores in the list *valid* all the continuous paths that are valid during this interval in *total order*, which is equal to the order of the temporal interval encoding.

These stored intervals are partitions of the intervals belonging to a continuous path in the sense of the greatest common divisor so that temporal intersections between multiple con-

tinuous paths can be pulled together. This means that a continuous path can appear in more than one *valid* list.

The δ_k tables are consulted at *snapshot* queries, where all valid nodes of a certain timestamp are asked e.g. the current semester program in on the homepage.

$\delta(S'0)$ -table

from	to	valid
0	Now	{0}

$\delta(S'1)$ -table

from	to	valid
0	Now	{1, 220}

$\delta(S'2)$ -table

from	to	valid
0	12	{2, 3, 34, 65, 221, 222}
13	Now	{2, 3, 34, 221, 222, 253}

$\delta(S'3)$ -table

from	to	valid
0	2	{4, 5, 8, 35, 36, 66, 67, 70, 223, 224}
3	12	{5, 8, 24, 35, 36, 66, 67, 70, 223, 224}
13	Now	{5, 8, 24, 35, 39, 223, 224, 254, 255}

$\delta(S'4)$ -table

from	to	valid
0	12	{6, 9, 37, 68, 71, 225}
13	Now	{6, 9, 40, 225, 256}

$\delta(S'5)$ -table

from	to	valid
0	12	{10, 72}
13	Now	{10}

Fig. 7 shows the δ for every depth (= summary node in the interval summary) in our teaching example

Second there is a *TCP table*⁴ for every summary node in the LCP summary, in which all the continuous paths with the corresponding path label are stored, like it has been summarized in the LCP summary. The tuple t inside the table contains the attributes *node*, *from*, *to* and *value*, where *node* is the temporal interval encoding of the continuous path leading to the requested node, *from* and *to* are the interval of the continuous path and *value* contains a final value if the node was an attribute

⁴ Again, these shortcuts are not explained in the article. TCP will probably stand for temporal continuous path.

node. The table is order by nodes which is equal to the total order.

The TCP tables are consulted in label based queries, where a selection of elements or attributes with a specific tag is requested or at operations concerning the final value.

Node (S1) TCP-table

node	from	to	value
0	0	Now	-

Node (S2) TCP-table

node	from	to	value
1	0	Now	-
220	0	Now	-

Node (S3) TCP-table

node	from	to	value
2	0	Now	Spring
221	0	Now	Autumn

Node (S4) TCP-table

node	from	to	value
3	0	Now	-
34	0	Now	-
65	0	12	-
222	0	Now	-
253	13	Now	-

Node (S5) TCP-table

node	from	to	value
4	0	2	Practical Information
24	3	Now	Technology
35	0	Now	Applied IT
66	0	12	IT: IIa
223	0	Now	IT: IIb
254	13	Now	IT: I

Node (S6) TCP-table

node	from	to	value
5	0	Now	-
36	0	12	-
39	13	Now	-
67	0	12	-
224	0	Now	-
255	13	Now	-

Node (S7) TCP-table

node	from	to	value
6	0	Now	Charles Xavier
37	0	12	Henry Jones Jr.
40	13	Now	Who
68	0	12	Abraham Van Helsing
225	0	Now	Hubert J. Farnsworth
256	13	Now	Abraham Van Helsing

Node (S8) TCP-table

node	from	to	value
8	0	Now	-
70	0	12	-

Node (S9) TCP-table

node	from	to	value
9	0	Now	-
71	0	12	-

Node (S10) TCP-table

node	from	to	value
10	0	Now	Victor Von Doom
72	0	12	Who

Fig. 8 shows the TCP table for every LCP summary nodes in our teaching example

4.5 Applying the temporal interval encoding

The presented indexing scheme is useful in finding an addressed node by a specific condition, but if there are several nodes involved in one query, their relationship among each is no longer evident e.g. to witch partOfYear does the lecture “Informatik I” belong to?

This is where the temporal interval encoding applies. Since a subnode has always a higher value than its supernode, we can isolate subtrees if we know the encoding value of the subtree’s root r and the encoding value of the successor n_s - in this case the node with the next higher encoding value on the equally or higher tree level, e.g. Fig. 5: the successor of node 1 is node 220; the successor of node 3 is node 34.

The requested successor is accessible in the TCP table of the summary node in which r belongs to. The next tuple in the table, which are ordered by node (= temporal interval encoding), is the requested successor. If there is no next entry, check the TCP table of the next higher summary node in the summary graph for the next entry that’s value is bigger than r .

All the nodes in the subtree of r will have a higher value than r , but a lower than n_s therefore the hierarchical order of the nodes can be restored by using the temporal interval encoding.

5 Querying

5.1 Query example

For querying a XML document there is the basic query language XPath that addresses nodes and subtrees of nodes by following the paths. Again, since this is temporal XML, there is an add-on delivered in order to use continuous paths instead of paths, so that the temporal level can be used and queried in his full potential. This extended version of XPath 2.0 is called TXPath where the input is a list of *tmap* tuples $[s, n, I]$ (a continuous path to node n^5 during the interval I associated to a summary node s) and the output is a list of valid continuous paths $[n, I]$.

In the presented article the authors break down all query expressions made by TXPath applied on TempIndex to be a sequence of six functions:

`getParent(Label)`, `getChildren(Label)`, `getDescendants(Label)`, `getAncestors(Label)`, where *Label* is a node label $\rightarrow$ these functions are using the LCP summary, the TCP tables and the temporal interval encoding are to access information inside the document graph.

`valFilter(valPred)` where *valPred* is a value predicate that is to be checked with the corresponding value entries in the TCP tables and `tempFilter(TempPred)` where *TempPred* is a temporal predicate that is to be checked in the corresponding δ table.

All preconditions for an explicit query are given, so let's consider the following question in the teaching example:

"A person passed the obligatory lecture "Practical Information Technology" some time ago. At the current time, the person wants to reach in their activity conformation. But meanwhile this course is no longer in the

teaching list. The question is whether the course is no longer valid or it has only been renamed and if so, which of the current courses is equivalent to it."

This person would query the temporal XML document *teaching* with the following TXPath expression:

```
teaching//lecture[name="Practical In-
formation Technology"]/name[@to =
"Now"]
```

This expression would be divided into a sequence of the six basic functions and each called function would use its corresponding data structures of the indexing scheme TempIndex to deliver a result for the next function:

```
List.add(root);
```

the tmap of the root is added to the *list*.

```
list{{S1, 0, [0,Now]}}
```

```
list = list.getDescendent("name");
```

Use the LCP summary graph to find all sub-nodes of S1 with the label "name": S3, S5, S7, S10 and overwrite *list* by all the tuples that are tmaped to this summary nodes (If the input tuple would not be the root of the summary, there would be needed a more complex search like discussed below at the `getChildren()` function).

```
list{{S3, 2, [0,Now]}; [S3, 221, [0,Now]}; [S5, 4,
[0,2]}; [S5, 24, [3,Now]}; [S5, 35, [0,Now]}; [S5,
66, [0, 12]}; [S5, 223, [0, Now]}; [S5, 254, [13,
Now]}; [S7, 6, [0, Now]}; [S7, 37, [0,12]}; [S7,
40, [13, Now]}; [S7, 68, [0,12]}; [S7, 225, [0,
Now]}; [S7, 256, [13, Now]}; [S10, 10, [0,
Now]}; [S10, 72, [0,12]}}
```

```
list = list.valFilter("Practical Informa-
tion Technology");
```

Check all the tuples in *list* if their value corresponds to the value predicate by checking the attribute *value* in the associated TCP table. Overwrite the *list* by all the succeeded tuples.

⁵ Consider that the continuous paths are encoded by the temporal interval encoding.

```
list{[S5, 4, [0,2]]}
```

```
list = list.getParent("lecture");
```

For every remaining tuple, use the summary graph to find the parental summary node labelled *lecture*, search in its TCP table for the tuple that's node value is the next smaller value to the node value of the input *list* tuple. Overwrite the *list* with all the parent tuples out of all for-loops.

```
list{[S4, 3, [0,Now]]}
```

```
list = list.getChildren("name");
```

For every remaining tuple *t*, use the TCP table of its summary node (here: S4), get the tuple with the next higher node value *t.node* and store it (successor *s* = [S4, 34, [0,Now]]). If there is none, check the LCP table of the parental summary node.

Return to the input *list* and use the summary graph to find the child summary node labelled with *name* (here: S5), search in its TCP table for all tuples that's node value is bigger than the parental one (*t.node*) but smaller than the one of the successor (*s.node*). Overwrite the *list* with all the child tuples out of all for-loops.

```
list{[S5, 4, [0,2]]; [S5, 24, [3,Now]]}
```

```
list = list.tempFilter("to = Now");
```

take all the δ table corresponding to the depth of the summary nodes stored in *list* (here only the δ_3 table), jump to the timestamp mentioned in the temporal predicate and check if the node value *t.node* of the *list* tuple are valid at the time. Overwrite *list* with the tuple that are valid.

```
list{[S5, 24, [3,Now]]}
```

RETURN only the continuous paths by cutting of the TSum attribute.

```
list{[24, [3,Now]]}
```

PRINT OUT the values of the result continuous paths: "Applied IT"

→ "Practical Information Technology" is indeed still in the teaching list, but with the new name "Applied IT"

6 Conculsion

To give a summary about the approach of F. Rizzolo and A. Vaisman there can be said, that it is possible to use XML even for mapping historical information. Therefore the XML elements and attributes need an add-on of a temporal tag that describes the temporal validity of the element/attribute.

This would be enough to be able of implementing temporal XML documents. The document can also be indexed and queried with standard structures and languages, but in order of granting a faster and a more useful access to the temporal component, all the data structures, indexes and query languages involved have been upgraded equivalently by the authors. The foundation of these new expansions is the adjustment from the usually used paths, leading to a corresponding element or attribute, to so called continuous paths, where the temporal add-on is included to the paths. A continuous path not only testifies where it is leading to but also when.

As a result F. Rizzolo and A. Vaisman created the new indexing scheme TempIndex in which the temporal component is included.

Experiments concerning the speed in different queries by using different indexing schemes (TempIndex – consist of temporal path summaries; ToXin – consist of non-temporal path summaries; DOM – consist of the representation of the document without any summaries), showed that indexing continuous paths is making the runtime smaller in every kind of query. Sometimes the difference is enormous.

7 The approach of F. Rizzolo and A. Vaisman

The authors of the discussed article provided not only an idea of solving the problem of XML by dealing with historical data, but presented the full packages of modelling and implementation, a new indexing scheme including additions to reuse existing data structures and an extensions for the query language XPath 2.0 → TXPath.

There are additional features in the article that hasn't been discussed in this text. For instance, the authors delivered algorithms for the most important functions used in their approach. A whole skipped part was dedicated to the finding and fixing of inconsistency in a XML document, since the conditions of temporal XML presented in their definition 3 can now be easily hurt – unlike in non-temporal XML⁶. Another part covered the topic of updating a XML document including dealing with insertions and deletions. All this sections complete the presented work of F. Rizzolo and A. Vaisman as a well-trough-out approach to the issue of temporal XML.

Unfortunately there are some fragments that aren't elaborated that well, like the successor function that is used multiple times but in different meanings or at least in different not indicated relation. Or the temporal interval encoding function, where the gap-function is a total mystery and even the example in which this function has been applied, isn't answering the question straightaway. Other information is simply missing, like the meaning of certain names or the namespace, which is necessary to create a temporal XML document and which is no longer available in the internet.

So, what looks like an elaborated package - ready for use, can't be applied by a user because of the lack of small information.

8 References

TEMPORAL XML: MODELING, INDEXING, AND QUERY PROCESSING. F. RIZZOLO, A. VAISMAN. INTERNATIONAL JOURNAL ON VERY LARGE DATA BASES (VLDBJ), 17(5), AUGUST 2008.

INDEXING TEMPORAL XML DOCUMENTS. A. MENDELZON, F. RIZZOLO, A. VAISMAN. 30TH INTERNATIONAL CONFERENCE ON VERY LARGE DATA BASES (VLDB). TORONTO, CANADA. SEPTEMBER 2004.

⁶ During the use of a temporal XML document, one can create for instance cycles by