TVX - Time and Versions in XML

Rodrigo Gasparoni Santos [1]
Nina Edelweiss [1]
Renata de Matos Galante [1]

Abstract: The proposed work explores the application of temporal and versioning management techniques to the content of XML documents. Recently, many methods have been proposed to address this problem. These methods differ in many aspects, such as which XML objects should be temporalized, thus possessing each one a set of advantages and limitations. We present the TVX model, whose goal is to combine these techniques into a single approach; its united use should supply great flexibility in the temporal treatment of XML documents. This will be accomplished through a high-level data model which can be easily implemented in XML format, along with XQuery-like query and data manipulation languages to handle the temporal XML documents. We also present a case study considering a historic query module over the text of the Brazilian Constitution, that demonstrates the employment of the TVX model in a real-life application.

1 Introduction

The use of temporal representation concepts has assumed an essential role in several database applications, because of its capacity to storage and manipulate the different states assumed by the data during the course of time. As the content of the databases evolves, the bitemporality concept provides access to past, present and future information. On the other hand, the versioning concept allows the existence of several alternatives for the evolution of database.

With the migration of such database applications to a Web environment, they adopt the XML language [1] as a standard format for representation and exchange of their internal data. Hence, such applications require a mechanism for the representation and manipulation of the history of the content of a XML document that goes through modifications along time. In spite of the existence of several proposals for the temporal extension of conventional data models (such as the many temporal extensions to the relational data model), as well as mapping strategies to storage XML documents in these models, they are not well-suited to handle generic semi-structured documents. In other words, the semi-structured nature of the XML documents requires the definition a new data model, capable to work with the bitemporality and versioning concepts in a semi-structured document.

[1]Instituto de Informática, UFRGS, Caixa Postal 15064 – 91501-970 Porto Alegre, RS, Brasil
{rgsantos, nina, galante@inf.ufrgs.br}

The interest for applying the concepts of temporal modeling to XML documents motivated several proposals in the last years, such as [2, 3, 4, 5, 6]. Such proposals tend to focus on the control of modifications to the content of the documents; only a few sketch mechanisms for dealing with schema evolution. In spite of that point in common, the proposals differ in many other aspects: some concentrate on registering the evolution through the storage of the several states that the data assumes [5]; others, through the operations that cause the transformations [6]; and others through a mix of information and references to previous states - [7] and [3] They also disagree in the definition of what type of information should be temporalized, but, mainly, they differ in the employment of temporal labels: some work with transaction time, others with validity time, and others just with the sequential numbering of the several states that the document assumes through time. Existing proposals will be discussed with further details in a latter section.

Even though these are, by themselves, powerful extensions to a conventional data model, they do not single-handedly cover all possible aspects in the evolution of an object. For instance, in a database that includes validity and transaction labels, the history of an object forms a linear timeline. The versioning concept [8] allows the storage of different alternatives of an object's history, thus creating the possibility for a branched timeline. Although the importance of schema evolution is undeniable, it won't be dealed with in the present work. It is intended, however, to establish a base for content evolution, in a way that allows its extension in future works to incorporate structural evolution.

The objective of the proposed work is to define a model for the evolution of the content of XML documents, unifying the dimensions of transaction time, validity time and creation of new versions. The united use of those concepts, treated separately in the studied proposals, provides great flexibility in the treatment of the documents evolution - versioning allows parallel lines in the evolution of the document; validity time, historical storage and the projection of future information; and transaction time, the recovery of past states of the database.

The remainder of this paper is organized as follows. Section 2 presents the pro-posed model, named *TVX*, standing for *Time and Versions in XML*. Sections 3 and 4 present two languages to be used along with the model, the first for querying and the second for updating the temporal XML documents. Section 5 contains a case study that was used to demonstrate the employment of the TVX model in a real-life application. Section 6 presents related works, comparing them with the present proposal, and Section 7 ends the paper with some conclusions and directions for future work.

2 The TVX Model

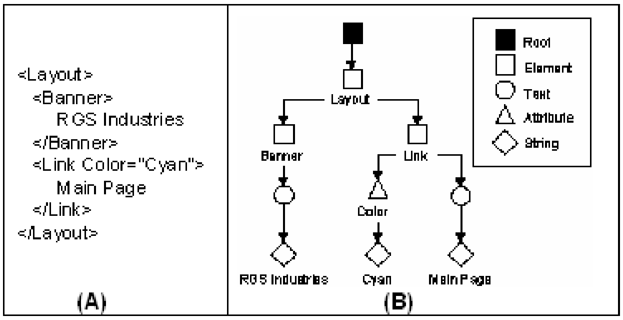
In this section, we begin our description of the TVX model (*Time and Versions in XML*) by formally defining the concept of a XML document according to the scope of this work. This definition will then be extended to incorporate temporal labels and version-ing information into the XML document. Through a high-level class model, we show how these extensions affect the XML objects and how they can be implemented in XML format. In the following discussion, we will use the terminology defined in [9], assuming that time moves forward linearly in discrete steps and that the special word now can be used to represent an open interval, associated to the ever-growing pre-sent time.

The definition of XML document that will be used is a simplification of the original proposal, which excludes the concepts of namespaces, processing instructions and comments. Those constructions were excluded to simplify the model and to allow it to focus on the content; in the same way, it won't be given any special treatment to ID, IDREF and IDREFS attributes; it should be pointed out, however, that the model can be extended to include such characteristics. To summarize, the objects of the language that will be considered are: elements, attributes and text nodes.

More formally, we view a XML document as a 6-tuple $< El, A, T, S, r, Ed >$, where El is a set of elements, A is a set of attributes, T is a set of text nodes, S is a set of string values, r is a distinguished object that points to the root of the document and Ed is a set of edges which connect the different XML objects such that the resulting graph is a tree. Each edge is a 2-tuple $< p, c >$, where p is the parent node and c the child node. The root element r cannot appear as a child in any edge, and must appear as a parent only once. Each edge must fall in one of the following categories:

- $p = r, c \in El$;
- $p \in El, c \in El$;
- $p \in El, c \in A$;
- $p \in El, c \in T$;
- $p \in A, c \in S$;
- $p \in T, c \in S$.

Elements and attributes are also given names, which must conform to the restriction that no two attributes under the same parent element can have the same name. Finally, there is a total order between the element and text children of an element node, such that no two text nodes appear consecutively in that order. Let us see an example of this view of an XML document. The tree in figure 1b is a graphical representation of this scheme applied to the small document seen in figure 1a, which represents the XML encoding for the layout of a Web page.



```
<Layout>
  <Banner>
     RGS Industries
  </Banner>
  <Link Color="Cyan">
     Main Page
  </Link>
</Layout>
```

Figura 1. Sample document (a) and corresponding modeling (b).

2.1 Modeling Temporal Labels

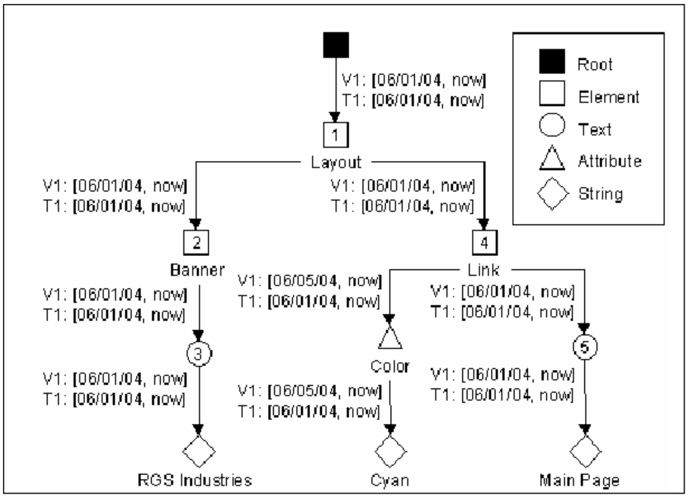
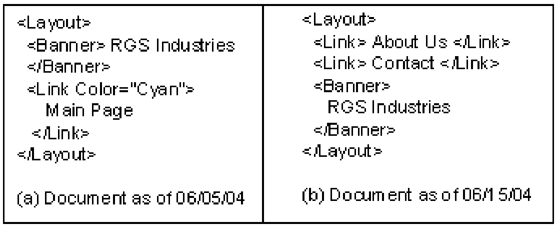


Figura 2. Original document with identifiers and temporal labels.

The XML model that we are considering will be extended to associate to each one of its objects a series of timestamps, to register the initial and final validity times - delimiting the period in which they model reality appropriately - as well as the initial and final transaction times - the period in which information was registered in the database. As mentioned in the introduction, the use of these labels allows one to go back to previous states and to register the lifespan of the stored objects in the real world. Element and text nodes also receive global and persistent identifiers, to indicate the correspondence between different states of the same object along the evolution of the document.

As an example, let us reconsider the small document seen in figure 1a, and assume that it was created at `06/01/04`. Figure 2 uses this example to show how the previous XML model can be extended to incorporate validity and transaction times as labels on the edges that link two objects, where the intervals refer to the child object of that edge. Each label is a series of pairs `(Vi, Ti)`, where `Vi` is a validity interval and `Ti` is the corresponding transaction interval. For instance, the labels for the `Layout` element indicate that it was registered at `06/01/04` (because that is the initial transaction time), it starts to be valid in the real world at that same moment (because the initial validity time has the same value), it should remain valid until further notice (because the final validity time is open) and that this series of labels is part of the current vision of the document (because the final transaction time is open). One may also notice that, despite being known to the database in `06/01/04`, the `Color` attribute only starts to be valid in `06/05/04`. Numbers within the geometric shapes represent the global identifiers for the element and text nodes.



```
<Layout>
  <Banner> RGS Industries
  </Banner>
  <Link Color="Cyan">
     Main Page
   </Link>
</Layout>

(a) Document as of 06/05/04
```

```
<Layout>
  <Link> About Us </Link>
  <Link> Contact </Link>
  <Banner>
    RGS Industries
  </Banner>
</Layout>

(b) Document as of 06/15/04
```

Figura 3. Example: two states of the same XML document.

Now, let us take a look at the two documents shown in figure 3 (the document in figure 3a is the same as in figure 1a); they actually represent two different states of the same document. For now, we will not worry about the fact that the `Banner` element went from being the first child of the `Layout` element to being its last child; the mechanism to properly model this change will be explained further ahead, in sub-section 2.3. Figure 3a shows the document at `06/05/04`. Later, at `06/10/04`, the document was edited to reflect the changes show in figure 3b, but this new state would only be valid starting from `06/15/04`. Figure 4 shows how the temporal labels can be used to reflect the changes applied to the sample document.

We can see that the `Color` attribute of the first `Link` element is no longer part of the current view of the document. The previous pair of labels, `(V1, T1)`, representing the time during which this object was believed to be valid indefinitely, is accurate just until `06/09/04` (represented by the final transaction time in the `T1` label). The current view, represented by the labels (`V2, T2`), is that this object is only valid from `06/05/04` to `06/14/04`. Accordingly, the old value for the text object within that same Link is re-placed with a new value ("`About Us`"); that change starts to represent reality at `06/15/04` (see correspondent validity time), and it is already know to the database at `06/10/04` (see correspondent transaction time). Even though the value of this text object has changed, the validity interval for the text node itself remained the same, because whether it was changed or not is irrelevant to the fact that there exists a text object at that position. More detailed explanations on how these temporal labels work together can be found at [10].

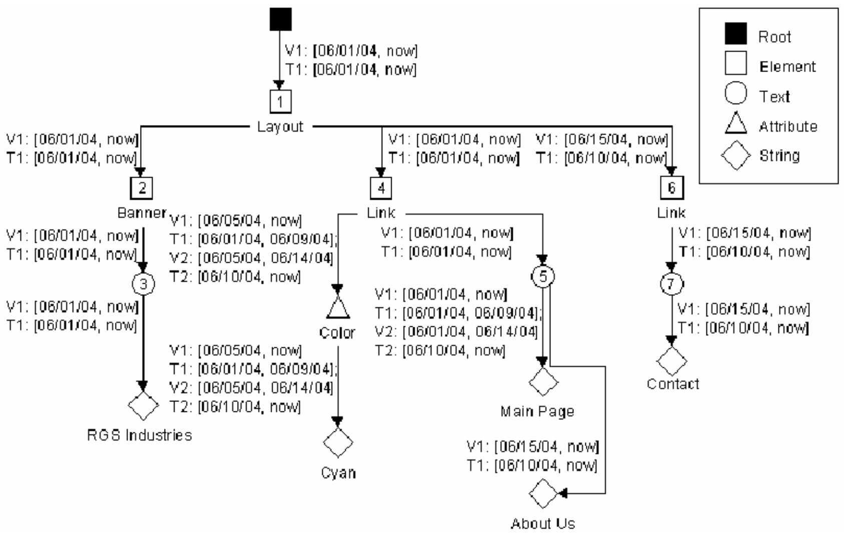


Figura 4. Updated document with identifiers and temporal labels.

Notice that with the conjoint use of the validity and transaction time labels, it is possible to identify four different states of knowledge:

- FROM `06/01/04` TO `06/04/04`. The current view of the document is as in figure 3a, excluding the `Color` attribute. It is already known, however, that this attribute will become valid at `06/05/04`.
- FROM `06/05/04` TO `06/09/04`. The current view of the document is as in figure 3a, and it is believed that it will remain in this way.
- FROM `06/10/04` TO `06/14/04`. The current view is still correspondent to figure 3a, but it is already known that, starting in `06/15/04`, the correct values will be as in figure 3b.

- STARTING AT `06/15/04`. The current view corresponds to figure 3b, but the knowledge about the previous states remains.

2.2 Modeling Versions

In addition to the timestamps for control of transaction and validity time, the proposed model also includes the possibility of defining versions of the document. The creation of a version is determined by the user, in case of a significant change in the document, and it is accomplished through replication and modification of the values of an existing version. A document always possesses a root version, from which there may be derived versions. The hierarchy of derivation of versions built according to these rules takes the form of a tree: each version possesses exactly one parent version (except for the root), and may possess one or more derived versions, which evolve independently from one another. The global identifiers indicate the correspondence of fragments of information between different versions.

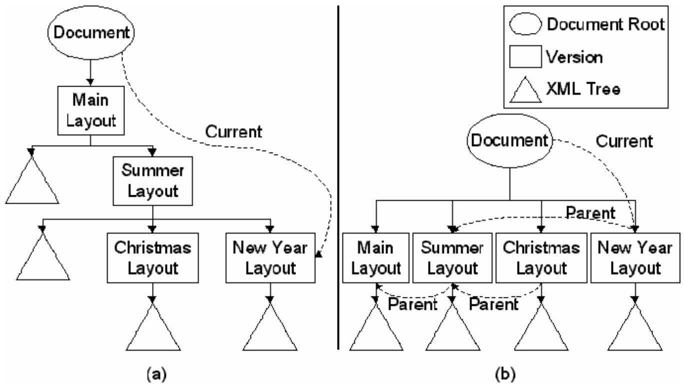


Figura 5. Versioning example - hierarchical structure (a) and flat structure (b).

For instance, imagine a company that wishes to change the layout of its homepage according to the time of the year (figure 5). Each layout is stored under the form of a XML document, such as the one in figure 1a. There is a main layout, which is used for most of the year - this will be the root version. When summer comes, the layout changes to a special `Summer` layout, which is a modified version of the main layout, thus being a derived version from the root version. During summer (if you live at the Southern Hemisphere), two special occasions occur, that cause other changes in the layout: `Christmas` and `New Year`; these will be new versions derived from the `Summer` version. Despite any similarities, each version has its own code, which must evolve separately from other versions. Therefore, each version node points to the root of its own version of the XML document; the lifetime of a version is considered to be the same of the root element it points to.

There may be objects common to many versions - for example, a banner at the top of the page with the name of the company. These can be modeled through repeated elements in the involved versions, all with the same global identifier. Also, there is the notion of a *current version*, which is simply the default target of update and query expressions when no particular version is specified. This means that all versions can be queried and/or updated at any given time, not just the current one.

Figure 5 shows two ways to model the relations between versions: through a hierarchical structure and through a flat structure. When encoding them in the form of XML documents, the hierarchical structure translates to nested versions, while the flat structure, in which the hierarchy is modeled through pointers, implies having each version completely isolated from the others within its own XML fragment. In this work, we chose to use the hierarchical structure, because then we can assign identifiers to the versions in such a way to speed-up the process of locating a given version within the XML document through its identifier, instead of scanning the whole file as it would be necessary with the flat structure. For instance, we could assign to the `New Year Layout` the identifier `1.1.2`, indicating that it is the second child of the first child of the root version, thus allowing one to find it by simply walking from the root.

2.3 The Complete Model

The complete TVX model can be described through the UML class diagram shown in figure 6. Each document can have an associated schema; that schema, if present, must be defined when the document is created, and once defined remains forever unchanged. The schema is, therefore, unalterable and unique for all the versions that ever come to be associated to that document. Each modification that takes place should maintain all states of the document, at any instant, coherent with the schema. If no schema is de-fined, then there is no restriction to the update operations that can be applied to the modeled document. The chosen format for representation of the schema is XML Schema, for three basic reasons: wide use and acceptance, superior flexibility in com-parison to DTD's and mainly for being a XML dialect, which makes it possible to in-corporate it directly into the document.

Each document has an initial version, from which it is possible to derive other versions, whose structures will be identical to the first one. Each version has a distinct element corresponding to the root of the XML document that it represents. Elements are codified in the `Element` class, with properties to register the name of the element and its identifier. Through inheritance, `Element` is involved in a `Validity` relationship that includes the several validity intervals corresponding to the lifespan of the referred `Element`. Each interval is defined through a series of timestamps, represented by the properties `IVT`, `FVT`, `ITT` and `FTT`,

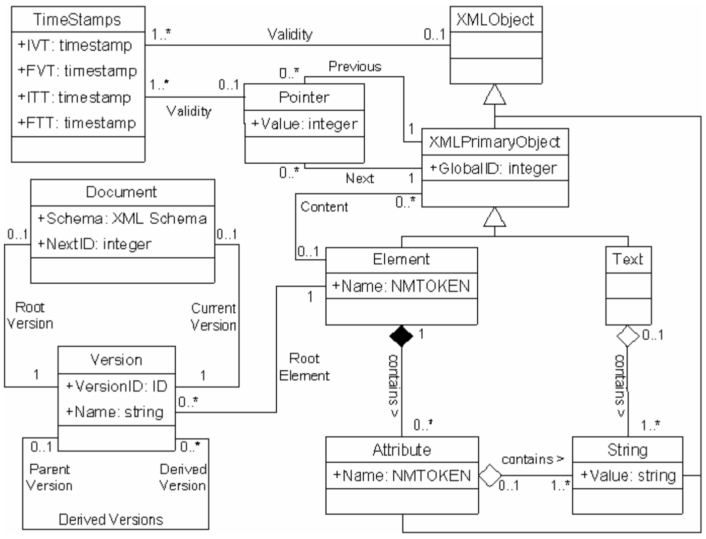


Figura 6. TVX class diagram.

which correspond, respectively, to the initial and final validity times and to the initial and final transaction times. The temporal labels of the root elements apply also to the versions to which they are subordinated.

For each `Element` there is also a (possibly empty) set of attributes which correspond to that element. Each attribute is modeled through an instance of the `Attribute` class, which also represents its lifetime through `Validity`. Throughout its existence, an attribute can assume several values, each one with its own validity interval; those are registered through the `String` class. Each `String` has a property that identifies its value, as well as timestamps to account for the temporal intervals associated to each one of the values assumed along time.

Finally, each element can contain subelements and text nodes; these are modeled through the `Content` relationship. The ordering between these children nodes is also subject to change, as it can be seen by the existence of timestamped `Previous` and `Next` pointers. Subelements have the same structure previously described; text nodes also have temporal labels for the object itself and for each one its values, built in the same way of the other properties. Each one of those objects presents the restriction that its life interval should be contained in the life interval of its immediate superior in the hierarchy. The `XMLObject` and `XMLPrimaryObject` classes are non-instantiable; they serve only to identify properties and relationships that are common to their specialized classes. Notice that the model does not define the granularity of the temporal labels, allowing it to vary according to the application.

To conclude the example, figure 7 shows how the evolving document in figure 3 can be coded with the proposed approach. For the sake of simplicity, the schema was omitted, and we give a detailed view only of the first `Link` element. Also, due to space constraints, versioning is not shown in this example (figure 3a and figure 3b are two different states of the same version), but a new version would simply be represented by another `Version` tag with a modified copy of the document under it, which would have the same structure seen in figure 7.

The understanding of most of the tags used in figure 7 is straightforward: elements, attributes and so on are modeled through tags with the same names. The temporal labels that appeared in the incoming edges in figures 2 and 4 are now timestamps under the `Validity` element, which is present under all the main tags; they still work in the same way as explained at the end of the subsection 2.1. Recall from the example from figures 3 and 4 that the way to properly model the change on the position of the `Banner` element was left unexplained. In figure 7 we can see, for every `Element` and `Text`, a pair of `Previous` and `Next` tags that work as pointers to the previous and next siblings of the represented object. The value contained in these pointers, which are also timestamped, is the global identifier of the pointed element/text; valid identifiers start from 1, so a value of zero indicates that there is no previous or next sibling. For the `Link` element shown in this example, we can see that it used to come after the `Banner(GlobalID = "2")` element, but after the promoted changes it has no predecessor, thus being the first child of its parent. Analogously, it used to be the last child of its parent, but now it is followed by another `Link` element - the one with the number 6 for its `GlobalID`.

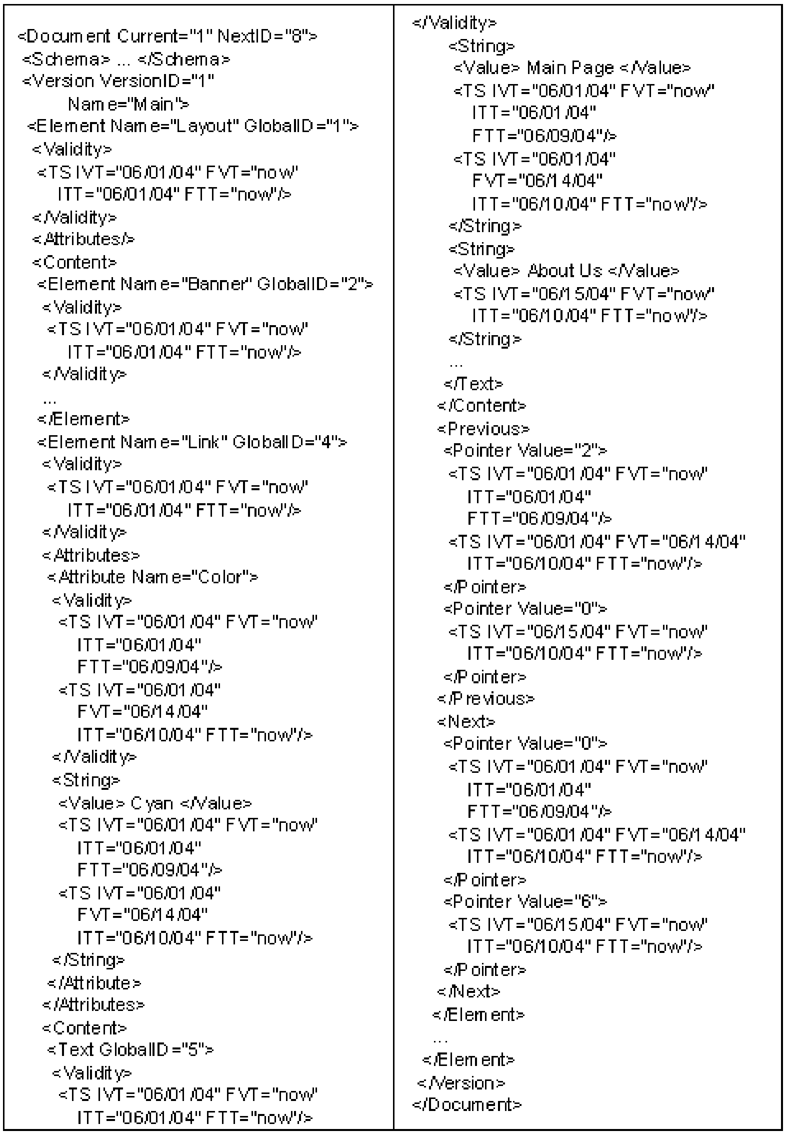
3 Query Language

The proposed query language is a modification to the standard XQuery [11], with modified XPath [12] expressions to filter the document according to a temporal predicate and to bind variables not only to elements, but also to attributes and text nodes. The additional constructions that may appear in the path expressions are as follows:

- In the layout document, the following query gives access to the root element of the Summer version:

  $FOR\ \$r\ in\ document(''layout.xml'')/tvx:version['Summer']/$
- To do a rollback and recover a past state of the database (say, at $06/08/04$) using the transaction time labels, the syntax is:

  $FOR\ \$r\ in\ document(''layout.xml'')/tvx:rollback['06/08/04']/$
- Likewise, the syntax to specify a given interval for validity time (say, from $06/07/04$ to $06/12/04$) is:



```
<Document Current="1" NextID="8">
 <Schema> ... </Schema>
 <Version VersionID="1"
        Name="Main">
  <Element Name="Layout" GlobalID="1">
   <Validity>
    <TS IVT="06/01/04" FVT="now"
        ITT="06/01/04" FTT="now"/>
   </Validity>
   <Attributes/>
   <Content>
    <Element Name="Banner" GlobalID="2">
     <Validity>
      <TS IVT="06/01/04" FVT="now"
          ITT="06/01/04" FTT="now"/>
     </Validity>
     ...
    </Element>
    <Element Name="Link" GlobalID="4">
     <Validity>
      <TS IVT="06/01/04" FVT="now"
          ITT="06/01/04" FTT="now"/>
     </Validity>
     <Attributes>
      <Attribute Name="Color">
       <Validity>
        <TS IVT="06/01/04" FVT="now"
            ITT="06/01/04"
            FTT="06/09/04"/>
        <TS IVT="06/01/04"
            FVT="06/14/04"
            ITT="06/10/04" FTT="now"/>
       </Validity>
       <String>
        <Value> Cyan </Value>
        <TS IVT="06/01/04" FVT="now"
            ITT="06/01/04"
            FTT="06/09/04"/>
        <TS IVT="06/01/04"
            FVT="06/14/04"
            ITT="06/10/04" FTT="now"/>
       </String>
      </Attribute>
     </Attributes>
     <Content>
      <Text GlobalID="5">
       <Validity>
        <TS IVT="06/01/04" FVT="now"
            ITT="06/01/04" FTT="now"/>
       </Validity>
        <String>
         <Value> Main Page </Value>
         <TS IVT="06/01/04" FVT="now"
             ITT="06/01/04"
             FTT="06/09/04"/>
         <TS IVT="06/01/04"
             FVT="06/14/04"
             ITT="06/10/04" FTT="now"/>
        </String>
        <String>
         <Value> About Us </Value>
         <TS IVT="06/15/04" FVT="now"
             ITT="06/10/04" FTT="now"/>
        </String>
        ...
       </Text>
      </Content>
      <Previous>
       <Pointer Value="2">
        <TS IVT="06/01/04" FVT="now"
            ITT="06/01/04"
            FTT="06/09/04"/>
        <TS IVT="06/01/04" FVT="06/14/04"
            ITT="06/10/04" FTT="now"/>
       </Pointer>
       <Pointer Value="0">
        <TS IVT="06/15/04" FVT="now"
            ITT="06/10/04" FTT="now"/>
       </Pointer>
      </Previous>
      <Next>
       <Pointer Value="0">
        <TS IVT="06/01/04" FVT="now"
            ITT="06/01/04"
            FTT="06/09/04"/>
        <TS IVT="06/01/04" FVT="06/14/04"
            ITT="06/10/04" FTT="now"/>
       </Pointer>
       <Pointer Value="6">
        <TS IVT="06/15/04" FVT="now"
            ITT="06/10/04" FTT="now"/>
       </Pointer>
      </Next>
     </Element>
     ...
   </Element>
  </Version>
</Document>
```

Figura 7. Example document adjusted to the TVX model.

$$FOR\ \$r\ in\ document(''layout.xml'')/tvx:validity['06/07/04','06/12/04']/$$

The above constructions can be combined in the same expression to allow more flexible queries. However; this is not mandatory. For instance, the current version is used in case the target version is not specified. The query can also include calls to functions specific to this query language (due to space constraints, we will not give a full list of these functions in this paper), which can be classified in three groups:

- FUNCTIONS THAT OPERATE ON THE HIERARCHY OF VERSIONS. Examples of such functions are: `current()`, to recover the identifier of the current version; `id()`, to recover the identifier of the root version; and `isparent(`version$_1$, version$_2$`)`, to test if version$_2$ is directly derived from version$_1$.
- FUNCTIONS THAT COMPARE TIME INSTANTS AND TIME INTERVALS. The definition of these functions was based on Allen's temporal relations [13]. Examples of such functions are: `equal(a, b)`, which returns whether parameters `a` and `b` contain the same time instant/interval; `precedes(a, b)`, to test if parameter `a` precedes parameter `b` in the timeline; and `intersect(a, b)`, which returns true if the intersection between `a` and `b` is a now-empty interval and false otherwise.
- FUNCTIONS THAT DEAL WITH MODEL-SPECIFIC DATA ON THE CONTENT OF THE XML DOCUMENTS. Examples of such functions are: `id(object)`, to return the global identifier of the referred object; `ftt(object)`, to return its final transaction time; and `snapshot(object)`, to return the XML fragment rooted in the referred object stripped of its temporal labels.

All the resources created for the query language are also available to the data manipulation language shown in the next section. Taking once more the document of the previous examples, the query in figure 8a returns a XML document containing the vision of the current version at `06/12/04` about the objects that would be valid between `06/07/04` and `06/13/04`. The result can be seen in figure 8b. The text content of the `Link` element shown in figure 8a does not appear in the result because, contrary to the `Link` itself and its attribute, none of its text values is valid during the entire specified range. The second `Link` element of the previous examples was left out entirely, because the `Link` element itself is only valid starting from `06/15/04`.

```
<Version Name="{vname(current())}"
         ID="vid(current())"> {
FOR $layout IN document("layout.xml")/
    tvx:rollback['06/12/04']/
    tvx:validity['06/07/04', '06/13/04']/layout
RETURN  snapshot($layout)  } </Version>
```

(a)

```
<Version Name="Main" ID="1">
 <Layout>
  <Link Color="Cyan"/>
  <Banner> RGS Industries </Banner>
 </Layout>
</Version>
```

(b)

Figura 8. (a) Query Example and (b) Query Result.

4 Data Manipulation Language

The proposed DML is also a modification to the standard XQuery, where, among other changes, the `RETURN` clause is replaced with an `UPDATE` clause. It is, in fact, a modification of an existing proposal [14], which was extended to incorporate temporal information into the update operations. The allowed operations that modify the content of the XML document are:

- INSERTION of new elements, attributes and text nodes. With exception of the attributes, there are variants to specify the position in which the new object should be inserted, with respect to an existing object. If no particular position is specified, the element/text is inserted as the last child of its parent.
- DELETION of an object and all its descendants.
- UPDATE of the content of attributes, text nodes and validity labels.
- MOVING an entire sub-tree to a new location inside the same XML document, with and without specification of the target position (cannot be applied to attributes; the target position may be a new location under the same parent node or to a different parent node). A move that does not specify the target position implicitly places the root of the XML subtree as the last child of its new parent.
- A new `FOR..LET..WHERE..UPDATE` expression (allows nested expressions, to work on multiple levels of the XML document with a single expression).

`UPDATE` clauses are followed by a binding to a variable and an ordered list of update commands. The bounded variable serves as an implicit target for all the update commands listed next to it. There are also operations for creating new versions and setting the current version, but these are not contained within an `UPDATE` clause. The syntax of the proposed operations is shown in figure 9. `FOR`, `LET` and `WHERE` clauses work in the same way as they do in standard XQuery, therefore their detailed syntaxes are omitted.

```
UpdateExpression ::= CreateDocument
                     | VersionManagement
                     | UpdateContent

CreateDocument ::= CREATE(RootName[, SchemaURI])
                   [FROM(DocURI, ivt,fvt)]
                   [AS UpdateContent]
VersionManagement ::= DERIVE(vid, NewName)
                      [AS UpdateContent] |
                      SETCURRENT(vid)
UpdateContent ::= FOR $binding IN XPath-expr, ...
                  LET $binding := XPath-expr, ...
                  WHERE predicate, ...
                  UPDATE $binding  UpdOp , UpdOp*
UpdOp ::= InsEl(name, ivt, fvt)
          | InsElBef($child, name, ivt, fvt)
          | InsElAft($child, name, ivt, fvt)
          | InsAt(name, str, ivt, fvt)
          | InsTxt(str, ivt, fvt)
          | InsTxtBef($child, str, ivt, fvt)
          | InsTxtAft($child, str, ivt, fvt)
          | Del($child, ivt)
          | Upd($child, str, ivt)
          | UpdIVT(ivt)
          | UpdFVT(fvt)
          | Mov($srcref, ivt)
          | MovBef($srcref, $child, ivt)
          | MovAft($srcref, $child, ivt)
          | UpdateContent
```

Figura 9. Grammar for update expressions.

Once again taking figure 3 as an example, the expression in figure 10 creates the document in figure 3a, and the expression in figure 11 transforms the document in figure 3a into the document appearing in figure 3b. To conclude this section, figure 12 gives an example of the derivation of a `Summer` version from the current `Main` version.

```
CREATE('Main') AS FOR $root IN / UPDATE $root  InsEl('Layout',
'06/01/04', 'now'),
               FOR $l IN $root/layout
               UPDATE $l  InsEl('Banner', '06/01/04', 'now'),
                           FOR $b IN $layout/banner
                           UPDATE $b  InsTxt('RGS Industries', '06/01/04', 'now') ,
                           InsEl('Link', '06/01/04', 'now'),
                           FOR $li IN $layout/link
                           UPDATE $li  InsAt('Color', 'Cyan', '06/05/04', 'now')
```

Figura 10. Example of a document creation expression.

```
FOR $la IN document("layout.xml")/tvx:version[1]/layout, $b IN
$la/banner UPDATE $la  FOR $li IN $la/link, $c IN $li/@color, $t IN
$li/text()
             UPDATE $li  Del($c, ’06/15/04’), Upd($t, ’About Us’, ’06/15/04’) ,
             InsEl(’Link’, ’06/15/04’, ’now’),
             FOR $li IN $la/link[2]
             UPDATE $li  InsTxt(’Contact’, ’06/15/04’, ’now’) ,
             Mov($t, ’06/15/04’)
```

Figura 11. Example of an update expression.

```
DERIVE(current(), ’Summer’) AS FOR $layout IN /layout UPDATE $layout
 InsEl(’Link’, ’12/20/04’, ’03/20/05’),
                FOR $link IN $layout/link[last()]
                UPDATE $link  InsTxt(’Click here for our Summer schedule’, ’12/20/04’,
                                      ’03/20/05’)
```

Figura 12. Example of a version derivation expression.

5 Case Study

The target application of this case study is a historic query module over the text of the *Brazilian Constitution*, accessible through the site of *Brazilian Treasury Department* (`http://www.receita.fazenda.gov.br`). The text of the Constitution is stored in XML format, currently through the *snapshot collection* method [15]. To briefly explain it, for each new state of the Constitution a new XML file is created, containing the text in its entirety, and this new file is then appended to the database. The new file contains both the new fragments and those that where left unchanged, thus resulting in a large amount of redundancy. The importance of temporal models for normative documents in XML format is better discussed in [16].

Figure 13 compares the employment of the snapshot collection method and of the TVX model to storage the text of the Brazilian Constitution as it changes through time (due to modifications promoted by Constitutional Amendments), regarding the size of the database. As a result, it is clear that the conversion of the XML documents from its original format to the TVX model format caused a great increase in the space required to storage the original state, when compared with the conventional technique. However, as the document suffered modifications, the difference was gradually lessened and eventually the TVX model became more advantageous than the other approach in terms of required space. One can also observe from figure 14 that this difference tends to increase with time. It is concluded, therefore, that the TVX model finds applicability in real situations, being possibly a better solution than others currently used.

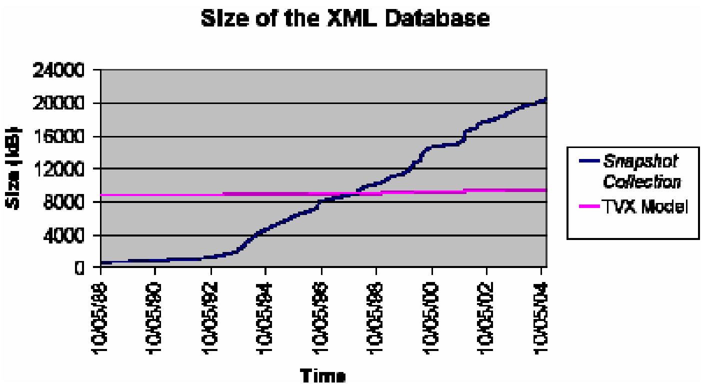
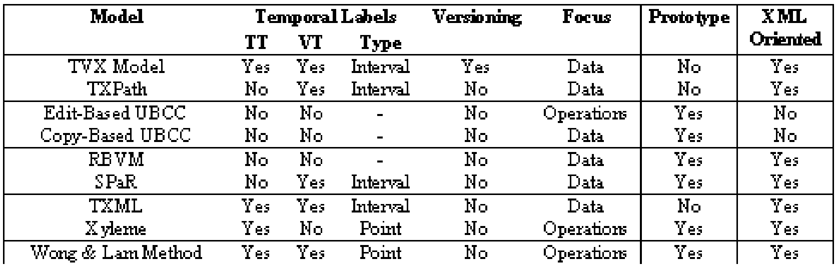


Figura 13. Space requirements for the TVX model and the Snapshot Collection method.

6 Related Work

Figure 14 synthesizes the differences and similarities between the TVX model and other existing proposals with similar goals. The comparison includes the TXPath model [17], the Edit-Based and Copy-based UBCC [3], the RBVM model [7], the SPaR method [2], the TXML language [4], the Xyleme method [18] and Wong & Lam's method [6].



Model	Temporal Labels			Versioning	Focus	Prototype	XML Oriented
	TT	VT	Type				
TVX Model	Yes	Yes	Interval	Yes	Data	No	Yes
TXPath	No	Yes	Interval	No	Data	No	Yes
Edit-Based UBCC	No	No	-	No	Operations	Yes	No
Copy-Based UBCC	No	No	-	No	Data	Yes	No
RBVM	No	No	-	No	Data	Yes	Yes
SPaR	No	Yes	Interval	No	Data	Yes	Yes
TXML	Yes	Yes	Interval	No	Data	No	Yes
Xyleme	Yes	No	Point	No	Operations	Yes	Yes
Wong & Lam Method	Yes	Yes	Point	No	Operations	Yes	Yes

Figura 14. Comparison between the TVX model and the studied proposals

The `TT` column indicates whether each model uses transaction time labels; similarly, the `VT` column indicates whether each model uses validity time labels. Depending on the model, each label may appear as either a single point in time or as a discrete in-terval. It is also shown which models support versioning, as well as which ones focus on registering evolution through the data itself and which ones register the operations that modify the data. The latter ones often require less space, but are more inefficient when recovering past states of the document. Finally, the last two columns indicate which models went beyond theory and were already implemented to some degree, and which models are XML oriented, thus taking advantage of specific characteristics of the XML language, instead of being developed for generic semi-structured documents.

The TVX model innovated in the sense that we added both transaction time and validity time to the conventional XML model, while the studied works (with the exception of the TXML language) employ only one (mostly transaction time), or sometimes neither, using instead only a sequential numbering of the many states the document assumes. Combined, these two types of temporal labels become much more powerful. To the best of our knowledge, this is also the first work to associate versioning characteristics (according to the definition presented in [8]) to XML documents. The term "version"does appear in other works, but with a different meaning, equivalent to the concept of "states"we used throughout the paper.

Compared to these works, however, the TVX model is at an early stage, with much left to be done. For instance, this paper did not cover implementation issues, such as storage organization and indexing mechanisms, as it is done in [2, 7, 3]. Also, contrary to [6], the query language does not provide direct support to refer explicitly to the operations that transform the documents (although it can be done by carefully considering the transaction time labels), such as querying which objects were deleted at a given moment, or to generate an edit-script that synthesizes the changes that occurred between two different states.

7 Summary and Future Work

The presented work showed a model capable of combining characteristics of many existing proposals, unifying temporal representation and versioning features in a single approach. This paper discussed only the logical organization of the data, and not its physical implementation. The proposed format for representing XML documents in the TVX model is also in XML, thus retaining all advantages inherent to this language, such as platform independence. Because it uses no mixed-content, it falls in the category of *Data Oriented XML Documents* [19]. As opposed to generic XML documents, these can be easily represented in both relational and XML native databases. Also, in the proposed languages, queries and update expressions are specified over the original structure defined for the document, and not in its internal representation.

This work is a first step in our research group to add temporal extensions to conventional XML documents. As such, many issues remain to be studied in future works:

- The study of physical implementation issues, such as physical organization of the data, index structures and query optimization.

- The adaptation of the TVX model to a multitemporal data model - i.e., one that can handle both temporal and static data in the same document.
- Currently, it is only possible to define versions of the XML document as a whole. One possible extension would be to reduce the granularity of the versioning scheme, allowing the definition of versions of fragments of the XML document.
- Most importantly, the inclusion of mechanisms to handle schema evolution.

Acknowledgments

The work of Rodrigo Gasparoni Santos is partially supported by the Brazilian National Research Council (CNPq).

Referências

[1] (W3C)., W.W.W.C.: Extensible markup language (xml). Available through WWW in `http://www.w3.org/XML/` (1998)

[2] Chien, S.Y., Tsotras, V.J., Zaniolo, C., Zhang, D.: Storing and querying multiversion xml documents using durable node numbers. In: 2nd Intl. Conf. on Web Information Systems Engineering (WISE). Volume 1., Kyoto, Japan, IEEE Computer Society (2001) 232–241

[3] Chien, S.Y., Tsotras, V.J., Zaniolo, C.: Efficient schemes for managing multiversion XML documents. The VLDB Journal **11** (2002) 332–353

[4] Manukyan, M.G.: Temporal data model. In: First East-European Symposium on Advances in Databases and Information Systems (ADBIS), St.-Petersburg, Nevsky Dialect (1997) 371–387

[5] Wang, F., Zaniolo, C.: Publishing and querying the histories of archived relational databases in XML. In: 4th Intl. Conf. on Web Information Systems Engineering (WISE), Rome, Italy, IEEE Computer Society (2003) 93–102

[6] Wong, R.K., Lam, N.: Efficient re-construction of document versions based on adaptive forward and backward change deltas. In: 4th Intl. Conf. on Database and Expert Systems Applications. Volume 2736 of Lecture Notes in Computer Science (LNCS)., Prague, Czech Republic, Springer (2003) 266–275

[7] Chien, S.Y., Tsotras, V.J., Zaniolo, C.: Efficient management of multiversion documents by object referencing. In: 27th Intl. Conf. on Very Large Data Bases (VLDB), Roma, Italy, Morgan Kaufmann (2001) 291–300

[8] Galante, R.M., Santos, C.S., Edelweiss, N., Moreira, Á.F.: Temporal and versioning model for schema evolution in object-oriented databases. Data & Knowledge Engineering **53** (2005) 98–128

[9] Jensen, C.S., et al.: The consensus glossary of temporal database concepts - february 1998 version. In: Temporal Databases: Research and Practice. Springer-Verlag (1998) 367–405

[10] Elmasri, R., Navathe, S.B.: Fundamentals of Database Systems. 3rd edn. Addison-Wesley Longman (2000)

[11] (W3C)., W.W.W.C.: Xquery 1.0: An xml query language. Available through WWW in `http://www.w3.org/TR/xquery/` (2001)

[12] (W3C)., W.W.W.C.: Xml path language (xpath). Available through WWW in `http://www.w3.org/TR/xpath/` (1999)

[13] Allen, J.F.: Maintaining knowledge about temporal intervals. Communications of the ACM (CACM) **26** (1983) 832–843

[14] Tatarinov, I., Ives, Z.G., Halevy, A.Y., Weld, D.S.: Updating XML. In: ACM SIGMOD Intl. Conf. on Management of Data, New York, NY, USA, ACM Press (2001) 413–424

[15] Chawathe, S.S., Abiteboul, S., Widom, J.: Representing and querying changes in semistructured data. In: 14th Intl. Conf. on Data Engineering (ICDE), Orlando, Florida, USA, IEEE Computer Society (1998) 4–13

[16] Grandi, F., Mandreoli, F., Tiberio, P., Bergonzini, M.: A temporal data model and management system for normative texts in XML format. In: 5th ACM CIKM Intl. Workshop on Web Information and Data Management (WIDM), New Orleans, Louisiana, USA, ACM (2003) 29–36

[17] Amagasa, T., Yoshikawa, M., Uemura, S.: A data model for temporal XML documents. In: 11th Intl. Conf. on Database and Expert Systems Applications (DEXA). Volume 1873 of Lecture Notes in Computer Science (LNCS)., London, UK, Springer (2000) 334–344

[18] Marian, A., Abiteboul, S., Cobena, G., Mignet, L.: Change-centric management of versions in an XML warehouse. In: 27th Intl. Conf. on Very Large Data Bases (VLDB), Roma, Italy, Morgan Kaufmann (2001) 581–590

[19] Bourret, R.: XML and databases. Available through WWW in `http://www.rpbourret.com/xml/XMLAndDatabases.htm` (2004)