# Schema Versioning in τXSchema-Based Multitemporal XML Repositories

Zouhaier Brahmia, Rafik Bouaziz
MIRACL laboratory, University of Sfax
Sfax, Tunisia
{zouhaier.brahmia, raf.bouaziz}@fsegs.rnu.tn

Fabio Grandi
DEIS, University of Bologna
Bologna, Italy
fabio.grandi@unibo.it

Barbara Oliboni
DI, University of Verona
Verona, Italy
barbara.oliboni@univr.it

***Abstract*—τXSchema [7] is a framework (a language and a suite of tools) for the creation and validation of time-varying XML documents. A τXSchema schema is composed of a conventional XML Schema document annotated with physical and logical annotations. All components of a τXSchema schema (i.e., conventional schema, logical annotations, and physical annotations) can change over time to reflect changes in user requirements or in reference world of the database. Since many applications need to keep track of both data and schema evolution, schema versioning has been long advocated to be the best solution to do this. In this paper, we deal with schema versioning in the τXSchema framework. More precisely, we propose a set of schema change primitives for the maintenance of logical and physical annotations and define their operational semantics.**

***Keywords-τXSchema; Schema versioning; XML; XML Schema; Temporal database***

## I. INTRODUCTION

Nowadays, XML [1] is becoming a standard for web documents. In database context, it is also used for representing (semi-)structured data. As for generic temporal databases [2], XML provides an excellent support for temporally grouped data models [3], which have been proposed as the most natural and effective representations of temporal information [4].

Both XML data and schemata tend to change over time for a multitude of reasons: changes in the real world or in the user's requirements, correction of mistakes in the initial design, etc.. Many applications (like banking, flight reservation, geographical information systems, e-business, e-government, etc.) require a complete maintenance of data and schema history for several reasons: avoiding loss of data after schema changes, maintenance of legacy data formatted according to past schemata, reuse of legacy applications, and auditing purposes. In order to keep track of both XML data and XML schema evolution over time, a temporal XML data model that supports schema versioning is required. Notice that schema versioning has long been advocated to be the more appropriate solution to support a complete data and schema history in databases [5,6].

When we started thinking about an approach for schema versioning in multitemporal XML repositories, our choices were as follows: (i) to have different levels of schema specifications, that is a level for the data structure and one or more levels for temporal dimensions, and (ii) to push the possible multitemporality one level higher. We have seen that "Which is the right way to consider XML documents sharing the same data structure and having different time dimensions" could be a good question. Hence, we had the problem to define the different levels we need, and to define the mappings between such levels.

After surveying the state of the art of (multi-)temporal XML data models supporting schema versioning, we conclude that the resulting overall framework could be not very dissimilar from the one introduced by Snodgrass and colleagues in [7,8,9], named τXSchema. This latter is an infrastructure, composed of an XML schema language and a suite of tools, for constructing and validating temporal XML documents under schema versioning. The τXSchema language extends the XML Schema language [10] to explicitly support time-varying XML documents. τXSchema has a three-level architecture for specifying a schema for time-varying data. The first level is for the **conventional schema** which is a standard XML Schema document that describes the structure of a standard XML document, without any temporal aspect. The second level is for the **logical annotations** of the conventional schema, which identify which elements can vary over time. The third level is for the **physical annotations** of the conventional schema, which describe where timestamps should be placed and how the time-varying aspects should be represented.

Finally, we were in front of two options:

- to extend the τXSchema approach;
- to propose a completely different approach.

We have chosen the first one, for the reasons which follow.

1) We came up with a similar requirement for having different levels for schema specification, so any alternative approach we could propose would not be so far from the τXSchema principles.

2) In case we decide to move away from τXSchema, we must then be very convincing in justifying our choice (e.g. by highlighting strong limitations of the τXSchema approach which we need to overcome).

3) The τXSchema approach is well known in the research community and thus it could be better to use it as a starting

point, instead of putting forward a brand new proposal.

4) In the τXSchema approach, there is room enough for extensions and, thus, we could define a set of schema changes and solve the semantics of change and change propagation problems for such operations on top of it.

In [7], the authors introduce τXSchema but did not discuss schema versioning. [8] and [9] deal with schema versioning in τXSchema: [8] focuses on cross-schema change validation and [9] extends it by discussing how to accommodate gaps in the existence time of an item, transaction semantics, and non-sequenced integrity constraints. All previous works on τXSchema focus on capturing a time-varying schema and validating documents against such a schema. They do not deal with how the schema changes are made, or what kinds of schema change operations can be supported. In this paper, we investigate these issues by proposing a set of schema change primitives that allow designers to change logical and physical annotations of a τXSchema schema. This set of primitives is complete, that is each annotation document can be generated starting from the empty document by applying a sequence of primitives, and for each annotation document a sequence of primitives exists for transforming it in the empty document. Moreover, this set is sound: i.e., each primitive applied to a consistent annotation document produces a consistent annotation document.

The remainder of this paper is organized as follows. Section 2 briefly describes the τXSchema framework. Section 3 presents our approach for versioning τXSchema logical and physical annotations. Section 4 introduces the schema change primitives that we propose for changing logical and physical annotations in the τXSchema framework. Section 5 surveys related works. Section 6 concludes the paper.

## II. THE τXSCHEMA FRAMEWORK

In this section, first we briefly present the τXSchema architecture (more details can be found in [11]), and then we provide a motivating example that illustrates the usage of τXSchema.

### *A. Architecture*

The τXSchema framework [7,8,9] allows a designer to create a temporal XML schema for (temporal) XML documents from a conventional schema (written in standard XML Schema language), logical annotations, and physical annotations. Figure 1 illustrates the architecture of τXSchema [11]. We note that only the components which are shaded in the figure are specific to an individual time-varying document and need to be supplied by a designer.

The designer starts with the conventional schema (box 3) which is a standard XML Schema document that describes the structure of the conventional document(s). A conventional document is a standard XML document that has no temporal aspects [11].

Then, the designer augments the conventional schema with logical annotations (box 5), which specify whether an element or attribute varies over valid time or transaction time, whether its lifetime is described as a continuous state or a single event, whether the item itself may appear at certain times (and not at others), and whether its content changes [11]. If no logical annotations are provided, the default logical annotation is that anything can change. However, once the designer has annotated the conventional schema, elements that are not described as time-varying are static and, thus, they must have the same content across every XML document in box 7.

After that, the designer augments the conventional schema with physical annotations (box 6), which specify the timestamp representation options chosen by the designer, such as where the timestamps are placed and their kind (e.g., valid time or transaction time) and the kind of representation adopted [11]. The location of timestamps is largely independent of which components vary over time. Timestamps can be located either on time-varying components (as specified by the logical annotations) or somewhere above such components. Two documents with the same logical information will look very different if we change the location of their physical timestamps. Changing an aspect of even one timestamp can make a big difference in the representation. τXSchema supplies a default set of physical annotations, which corresponds to timestamp the root element with valid and transaction times. However, adding them can lead to more compact representations.

Logical and physical annotations are orthogonal and are independently maintained, although they are stored together in a single document related to the conventional schema, which is a standard XML document named the annotation document. The schema for the logical and physical annotations is given by ASchema (box 2).

By separating the conventional schema, logical annotations, and physical annotations, the three-level architecture of τXSchema guarantees data independence and allows each component to be changed independently.

Finally, the designer creates the temporal schema document (box 4) in order to provide the linking information between the conventional schema, logical annotations, and physical annotations. The temporal schema is a standard XML document that ties the conventional schema, logical annotations, and physical annotations together [11]. The temporal schema in the τXSchema environment is the logical equivalent of the conventional XML Schema in the non-temporal XML environment. This document contains sub-elements that associate a series of conventional schema definitions with logical and physical annotations, along with the time span during which the association was in effect. The schema for the temporal schema document is TSSchema (box 1).

Notice that, whereas the introduction of TSSchema (box 1) and ASchema (box 2) is due to Snodgrass and colleagues, XML Schema (box 0) is the standard endorsed by the W3C.

The temporal schema document (box 4) is processed by the temporal validator τXMLLINT in order to ensure that the logical and physical annotations are (i) valid with respect to ASchema, and (ii) consistent with the conventional schema. τXMLLINT reports whether the temporal schema document is valid or invalid.

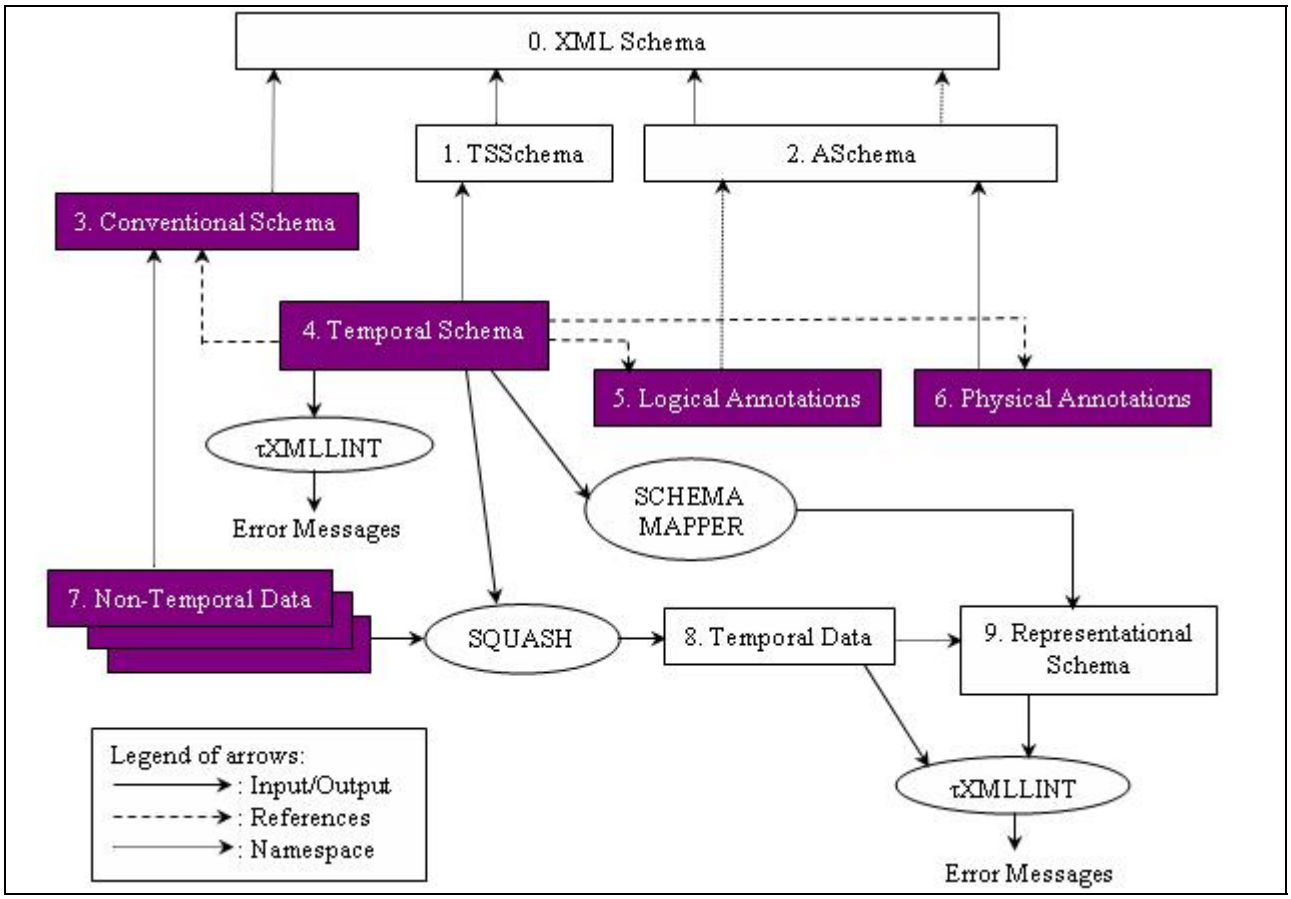


Figure 1. Architecture of τXSchema.

Once the annotations are found to be consistent, the Schema Mapper generates the representational schema (box 9) from the temporal schema (i.e., from the conventional schema and the logical and physical annotations). The representational schema becomes the schema for temporal data (box 8). Temporal data can be automatically created from the non-temporal data (box 7) and the temporal schema (box 4), using the Squash tool. Moreover, temporal data are validated against the representational schema through τXMLLINT which reports whether the temporal data document is valid or invalid.

### *B. Motivating Example*

Assume that a new bank requires an XML repository for storing data and schema of customer accounts. On February 1, 2010, the designer creates the first version of the conventional schema shown in Figure 2. Each account in this bank is described by its number, the name of its owner, its opening date, its type, and its balance. Then, the designer annotates this first version of the conventional schema with some logical and physical annotations. As to logical annotations, suppose that he/she decides to make the content of the <Balance> element varying in transaction-time (in order to keep the history of the balance of each account along transaction time). As to physical annotations, suppose that he/she chooses to add a transaction-time physical timestamp to the element <Account> (i.e., whenever any element below <Account> changes, the entire <Account> element is repeated).

The first version of the annotation document related to the conventional schema of the bank is shown in Figure 3. Finally, the designer creates the temporal schema to define the links between the conventional schema and the annotation document, as shown in Figure 4.

```
<?xml version="1.0" encoding="UTF-8"?>
<schema xmlns="http://www.w3.org/2001/XMLSchema">
 <element name="Bank">
  <complexType>
   <sequence>
    <element ref="Account"/>
     </sequence>
  </complexType>
 </element>
 <element name="Account">
  <complexType>
   <sequence>
    <element name="OwnerName" type="string"/>
        <element name="OpeningDate" type="date"/>
    <element name="Type" type="string"/>
        <element name="Balance" type="float"/>
     </sequence>
   <attribute name="Number"
        type="nonNegativeInteger" use=”required”/>
   </complexType>
  </element>
</schema>
```

Figure 2. First version of the conventional schema (Bank_V1.xsd), on February 1, 2010.

```
<?xml version="1.0" encoding="UTF-8"?>
<annotationSet xmlns="http://www.cs.arizona.edu/tau/tauXSchema/
                        ASchema">
 <logical>
  <item target="/Bank/Account/Balance">
   <transactionTime kind="state" content="varying" existence="constant"/>
   <itemIdentifier name="balanceID" timeDimension="transactionTime">
    <field path="."/>
   </itemIdentifier>
    </item>
 </logical>
<physical>
  <stamp target="/Bank/Account"
       dataInclusion=”expandedVersion”>
   <stampKind timeDimension="transactionTime" stampBounds="extent"/>
    </stamp>
 </physical>
</annotationSet>
```

Figure 3. First version of the annotation document (BankAnnotations_V1.xml), on February 1, 2010.

```
<?xml version="1.0" encoding="UTF-8"?>
<temporalSchema xmlns="http://www.cs.arizona.edu/tau/tauXSchema/
                        TSSchema">
   <conventionalSchema>
      <sliceSequence>
          <slice location="Bank_V1.xsd" begin="2010-02-01" />
      </sliceSequence>
   </conventionalSchema>
   <annotationSet>
      <sliceSequence>
          <slice location="BankAnnotations_V1.xml"begin="2010-02-01" />
      </sliceSequence>
   </annotationSet>
</temporalSchema>
```

Figure 4. Temporal schema (BankTemporalSchema.xml) on February 1, 2010.

## III. VERSIONING OF τXSCHEMA LOGICAL AND PHYSICAL ANNOTATIONS

In this section, we briefly describe how τXSchema logical and physical annotations are versioned in our approach.

The first step of a schema versioning sequence is the creation of the first schema version: the designer creates a conventional XML-Schema document (i.e., an XSD file) annotated with some logical and physical annotations in an independent document (which is stored as an XML file), through a graphical interface. Moreover, he/she creates the temporal schema (also stored as an XML file) that ties together the conventional schema and the annotations.

In further steps of the versioning sequence, when necessary, the designer can independently change the conventional schema, the logical annotations or the physical annotations.

Changing the conventional schema leads to a new version of it. Similarly, changing a logical or a physical annotation leads to a new version of the whole annotation document. Therefore, the temporal schema is updated after each change of the conventional schema or of the annotation document. In this paper, we do not deal with changes of the conventional schema.

Schema change operations performed by the designer are high-level, since they are usually conceived having in mind high-level real-world object properties. Each of these high-level schema change operations is then mapped onto a sequence of low-level schema change operations (or primitive changes) by the schema change processor to be implemented.

In this paper, we investigate primitive changes of logical and physical annotations and not high-level changes. In fact, each high-level change can be expressed as a sequence of primitive changes. Thus, the consistency of the resulting annotation document is always guaranteed.

Primitive changes are applied to the annotation document which contains all logical and physical annotations related to the conventional schema. The structure of an annotation document, named ASchema, was described in detail by Currim et al. in [11].

## IV. PRIMITIVE CHANGES FOR τXSCHEMA LOGICAL AND PHYSICAL ANNOTATIONS

In this section, we first present our design choices, then we describe primitive changes acting on logical and physical annotations in τXSchema, and finally we give an illustrative example. We have individuated primitive operations (i.e., non-further decomposable in terms of the other ones) which make up a complete set of changes (i.e., such that any possible complex change can be defined via a combination/sequence of them).

### *A. Design Choices*

- All primitives must work on a valid Annotation Document (AD), that is have a valid AD as input and produce a valid AD as output.
- All primitives need to work on an XML file storing the AD, whose name must be supplied as argument.
- Stamp and Item elements are identified by their “target” attribute; a “stamp path” (“item path”) argument does not seem to be necessary; also because the order of stamp (item) elements within the physical (logical) container is irrelevant.
- Primitives adding elements with possibly optional attributes have the values for all the attributes as arguments; empty places in the argument list stand for unspecified optional attributes.
- We use Add…/Change… primitives for all elements which have multiple occurrences (e.g. “include”); a single Set… primitive is used for adding/changing elements/attributes with occurrences <=1 (e.g. “dataInclusion”).
- Elements without attributes which are just containers for sets of sub-elements (e.g. <logical/> and <physical/>, <orderBy/>) can be managed by the primitives concerning the sub-elements, without specific primitives acting on them (i.e., the container is created when the first sub-element is created and is deleted when the last sub-element is deleted).

- For all primitives, arguments which are used to identify the object on which the primitive works are in the first place of the argument list.
- The definition of include/defaultTimeFormat are exactly the same both for <physical/> and for <logical/>. In case, the same primitives acting on include/defaultTimeFormat can be used either for <physical/> or <logical/> elements (with an argument toWhat to choose between them).

The list of primitives which follows are the applications of the design choices presented above. We have organized them into four categories: (i) primitives acting on the AD as a whole (in the sub-section B), (ii) primitives that are common to the logical and to the physical annotations (in the sub-section C), (iii) primitives that are specific to the physical annotations (in the sub-section D), and (iv) primitives that are specific to the logical annotations (in the sub-section E).

For each primitive change, we describe its arguments and its operational semantics. Obviously, each primitive change has an effect on the AD. Due to space limitations, we do not present in this paper the effects of all primitive changes. We give only the effect of some chosen primitive changes. More detailed description of the primitives' operational semantics and their effects on the AD can be found in [12].

### *B. Primitive Changes Acting on the Whole Annotation Document*

We have only two primitive changes; we list them in Table 1. In the following, we choose to present only the effect of the CreateAnnotationDocument (AD.xml) primitive change. The contents of the AD.xml file after the application of such a primitive is as follows:

AD.xml:

```
<?xml version="1.0" encoding="UTF-8"?>
<annotationSet
        xmlns="http://www.cs.arizona.edu/tau/tauXSchema/ASchema/">
```

### *C. Primitive Changes Common to the Logical and to the Physical Annotations*

These primitive changes can be applied either to the <logical/> or to the <physical/> container. We have defined six primitive changes; we list them in Table 2. Here, we choose to present only the effects of the following primitive changes: AddInclude and AddDefaultTimeFormat.

TABLE I. PRIMITIVE CHANGES ACTING ON THE WHOLE ANNOTATION DOCUMENT

| Primitive change | Description |
|---|---|
| CreateAnnotationDocument( AD.xml) | It produces a valid empty AD. According to the second design choice, the argument is the name of the XML file where the new AD is stored. |
| DropAnnotationDocument( AD.xml) | It removes the AD.xml file from disk, with the constraint that the argument represents an empty AD (i.e. like the one above initially created by CreateAnnotationDocument). Any other contents must have been removed before. |

TABLE II. PRIMITIVE CHANGES COMMON TO THE LOGICAL AND TO THE PHYSICAL ANNOTATIONS

| Primitive change | Description |
|---|---|
| AddInclude(AD.xml,toWhat, annotationLocation) | Adds the <include/> element with specified annotationLocation to the fromWhat (i.e. <physical/> or <logical/>) container. Notice that any number of <include/> elements can be added, hence the annotationLocation is generally needed to distinguish between them. |
| DeleteInclude(AD.xml, fromWhat, annotationLocation) | Removes the <include/> element with specified annotationLocation from the fromWhat (i.e. <physical/> or <logical/>) container. |
| ChangeInclude(AD.xml,inWhat, oldannotationLocation, newannotationLocation) | Changes the value of the attribute annotationLocation of the <include/> element with specified oldannotationLocation in the inWhat (i.e. <physical/> or <logical/>) container. |
| AddDefaultTimeFormat(AD.xml, toWhat,plugin, granularity,calendar, properties,valueSchema) | Adds the <defaultTimeFormat/> element with specified plugin, granularity, calendar, properties, and valueSchema to the fromWhat (i.e. <physical/> or <logical/>) container. |
| DeleteDefaultTimeFormat(AD.xml, fromWhat) | Removes the <defaultTimeFormat/> element from the fromWhat (i.e. <physical/> or <logical/>) container. |
| SetDefaultTimeFormat(AD.xml, inWhat,plugin, granularity,calendar, properties,valueSchema) | Changes plugin, granularity, calendar, properties, or valueSchema attributes of the <defaultTimeFormat/> element in the inWhat (i.e. <physical/> or <logical/>) container. |

The effect of the AddInclude(AD.xml, toWhat, annotationLocation) primitive, that is the contents of the AD.xml file after its application, is as follows:

If **toWhat** = physical

AD.xml:

```
<?xml version="1.0" encoding="UTF-8"?>
<annotationSet
        xmlns="http://www.cs.arizona.edu/tau/tauXSchema/ASchema">
<physical>
  <include annotationLocation="annotationLocation"/>
</physical>
</annotationSet>
```

Else:

AD.xml:

```
<?xml version="1.0" encoding="UTF-8"?>
<annotationSet
        xmlns="http://www.cs.arizona.edu/tau/tauXSchema/ASchema">
<logical>
  <include annotationLocation="annotationLocation"/>
</logical>
</annotationSet>
```

The effect of the AddDefaultTimeFormat(AD.xml, toWhat, plugin, granularity, calendar, properties, valueSchema) primitive, that is the contents of the AD.xml file after its application, is as follows:

If **toWhat** = physical

AD.xml:

```
<?xml version="1.0" encoding="UTF-8"?>
```

```
<annotationSet
        xmlns="http://www.cs.arizona.edu/tau/tauXSchema/ASchema">
<physical>
  <defaultTimeFormat>
    <format plugin="plugin" granularity ="granularity"
            calendar="calendar" properties="properties"
            valueSchema="valueSchema"/>
  </defaultTimeFormat>
</physical>
</annotationSet>
```

Else:

AD.xml:

```
<?xml version="1.0" encoding="UTF-8"?>
<annotationSet
        xmlns="http://www.cs.arizona.edu/tau/tauXSchema/ASchema">
<logical>
  <defaultTimeFormat>
    <format plugin="plugin" granularity ="granularity"
            calendar="calendar" properties="properties"
            valueSchema="valueSchema"/>
  </defaultTimeFormat>
</logical>
</annotationSet>
```

### *D. Primitive Changes Related to the Physical Annotations*

These primitive changes can be applied only to the <physical/> container. We have defined nine primitive changes; we list them in Table 3. Here, we choose to present only the effects of the following primitive changes: AddStamp and AddOrderByFieldToStamp.

TABLE III. PRIMITIVE CHANGES RELATED TO THE PHYSICAL ANNOTATIONS

| Primitive change | Description |
|---|---|
| AddStamp(AD.xml, stampTarget, stampDataInclusion, stampKindTimeDimension, stampKindStampBounds) | Adds the <stamp/> element with specified stampTarget, stampDataInclusion, stampKindTimeDimension, and stampKindStampBounds to the <physical/> container, where the three last arguments are optional. Possible values of some arguments: - stampDataInclusion: one of expandedEntity, referencedEntity, expandedVersion, or referencedVersion. - stampKindTimeDimension: one of validTime, transactionTime, or bitemporal. - stampKindStampBounds: either step or extent. |
| DeleteStamp(AD.xml,stampTarget) | Removes the <stamp/> element with specified stampTarget from the <physical/> container. |
| SetDataInclusionInStamp(AD.xml, stampTarget, stampDataInclusion) | Introduces or changes dataInclusion attribute of the <stamp/> element with specified stampTarget in the <physical/> container. |
| SetStampKindInStamp(AD.xml, stampTarget, stampKindTimeDimension, stampKindStampBounds) | Introduces or changes timeDimension and/or stampBounds attributes of the <stampKind/> element of the <stamp/> element with specified stampTarget in the <physical/> container. |
| SetFormatInStampKind(AD.xml, stampTarget, stampPlugin, stampGranularity, stampCalendar, stampProperties, stampValueSchema) | Adds (or changes) the <format/> element with specified stampPlugin, stampGranularity, stampCalendar, stampProperties, and stampValueSchema to the <stampKind/> element of the <stamp/> element with specified stampTarget in the <physical/> container. |
| DeleteFormatFromStampKind( AD.xml, stampTarget) | Removes the <format/> element from the <stampKind/> element of the <stamp/> element with specified stampTarget in the <physical/> container. |
| AddOrderByFieldToStamp( AD.xml, stampTarget, newOrderByField) | Adds a <field/> element having the value newOrderByField to the <orderBy/> element of the <stamp/> element with specified stampTarget in the <physical/> container. |
| DeleteOrderByFieldFromStamp( AD.xml, stampTarget, OrderByField) | Removes the <field/> element having the value OrderByField from the <orderBy/> element of the <stamp/> element with specified stampTarget in the <physical/> container. When the last <field/> is removed, also the <orderBy/> container is removed. Here, OrderByField is the value of the attribute dimension of a <time/> element or the value of a <target/> element in a <field/> element of the <orderBy/> element. |
| ChangeOrderByFieldInStamp( AD.xml, stampTarget, oldOrderByField, newOrderByField) | Changes the <field/> element having the value oldOrderByField to the value newOrderByField, in the <orderBy/> element of the <stamp/> element with specified stampTarget in the <physical/> container. |

The effect of the AddStamp(AD.xml, stampTarget, stampDataInclusion, stampKindTimeDimension, stampKindStampBounds) primitive, that is the contents of the AD.xml file after its application, is as follows:

AD.xml:

```
<?xml version="1.0" encoding="UTF-8"?>
<annotationSet
        xmlns="http://www.cs.arizona.edu/tau/tauXSchema/ASchema">
<physical>
  <stamp target="stampTarget" dataInclusion="stampDataInclusion">
    <stampKind timeDimension="stampKindTimeDimension"
               stampBounds="stampKindStampBounds"/>
  </stamp>
</physical>
</annotationSet>
```

The effect of the AddOrderByFieldToStamp(AD.xml, stampTarget, newOrderByField) primitive, that is the contents of the AD.xml file after its application, is as follows:

If **newOrderByField** in {validTime, transactionTime}

AD.xml:

```
<?xml version="1.0" encoding="UTF-8"?>
<annotationSet
        xmlns="http://www.cs.arizona.edu/tau/tauXSchema/ASchema">
<physical>
  <stamp target="stampTarget" dataInclusion="stampDataInclusion">
    <stampKind timeDimension="stampKindTimeDimension"
               stampBounds="stampKindStampBounds"/>
    <format plugin="stampPlugin" granularity="stampGranularity"
            calendar="stampCalendar" properties="stampProperties"
```

```
        valueSchema="stampValueSchema" />
    <orderBy>
      <field>
        <time dimension="newOrderByField"/>
      </field>
    </orderBy>
  </stamp>
</physical>
</annotationSet>
```

Else:

AD.xml:

```
<?xml version="1.0" encoding="UTF-8"?>
<annotationSet
        xmlns="http://www.cs.arizona.edu/tau/tauXSchema/ASchema">
<physical>
  <stamp target="stampTarget" dataInclusion="stampDataInclusion">
    <stampKind timeDimension="stampKindTimeDimension"
                stampBounds="stampKindStampBounds"/>
    <format plugin="stampPlugin" granularity="stampGranularity"
            calendar="stampCalendar" properties="stampProperties"
            valueSchema="stampValueSchema" />
    <orderBy>
      <field>
        <target>newOrderByField</target>
      </field>
    </orderBy>
  </stamp>
</physical>
</annotationSet>
```

### *E. Primitive Changes Related to the Logical Annotations*

These primitive changes can be applied only to the <logical/> container. We have identified forty one primitive changes; we list them in Table 4. Here, we choose to present only the effects of the following primitive changes: AddItem, AddValidTimeToItem, and AddContentVaryingApplicabilityToValidTimeInItem.

TABLE IV. PRIMITIVE CHANGES RELATED TO THE LOGICAL ANNOTATIONS

| Primitive change | Description |
|---|---|
| AddItem(AD.xml,itemTarget) | Adds the <item/> element with specified itemTarget to the <logical/> container. |
| DeleteItem(AD.xml,itemTarget) | Removes the <item/> element with specified itemTarget from the <logical/> container. |
| AddValidTimeToItem(AD.xml,<br>itemTarget,<br>validTimeKind,<br>validTimeContent,<br>validTimeExistence) | Adds the <validTime/> element with specified validTimeKind, validTimeContent, and validTimeExistence to the <item/> element with specified itemTarget in the <logical/> container, where the three last arguments are optional.<br>Possible values of some arguments:<br>- validTimeKind: either state or event.<br>- validTimeContent: either constant or varying.<br>- validTimeExistence: one of constant, varyingWithGaps, varyingWithoutGaps. |
| DeleteValidTimeFromItem(<br>AD.xml,<br>itemTarget) | Removes the <validTime/> element from the <item/> element with specified itemTarget in the <logical/> container. |
| SetValidTimeInItem(AD.xml,<br>itemTarget,<br>validTimeKind,<br>validTimeContent,<br>validTimeExistence) | Changes the value of the kind attribute and/or the value of the content attribute and/or the value of the existence attribute of the <validTime/> element of the <item/> element with specified itemTarget in the <logical/> container. |
| AddContentVaryingApplicabilityTo-ValidTimeInItem(<br>AD.xml,<br>itemTarget,<br>contentVaryingApplicabilityBegin,<br>contentVaryingApplicabilityEnd) | Adds the <contentVaryingApplicability /> element with specified contentVaryingApplicability-Begin and contentVaryingApplicability-End to the <validTime/> element of the <item/> element with specified itemTarget in the <logical/> container. This primitive is not applicable to a <validTime/> element with an attribute content having the value constant. |
| DeleteContentVaryingApplicability-FromValidTimeInItem(<br>AD.xml,<br>itemTarget,<br>contentVaryingApplicabilityBegin,<br>contentVaryingApplicabilityEnd) | Removes the <contentVaryingApplicability /> element with specified oldcontentVaryingApplicabi-lityBegin and oldcontentVaryingApplicabi-lityEnd from the <validTime/> element of the <item/> element with specified itemTarget in the <logical/> container. |
| ChangeContentVaryingApplicabilityInVa lidTimeInItem(<br>AD.xml,<br>itemTarget,<br>oldcontentVaryingApplicabilityBegin,<br>oldcontentVaryingApplicabilityEnd,<br>newcontentVaryingApplicabilityBegin,<br>newcontentVaryingApplicabilityEnd) | Changes the value of the begin attribute and/or the value of the end attribute of the <contentVaryingApplicability /> element with specified oldcontentVaryingApplicabi-lityBegin and oldcontentVaryingApplicabi-lityEnd in the <validTime/> element of the <item/> element with specified itemTarget in the <logical/> container. |
| SetMaximalExistenceInValidTimeInItem (AD.xml,<br>itemTarget,<br>MaximalExistenceBegin,<br>MaximalExistenceEnd) | Adds or changes the <maximalExistence/> element with specified maximalExistenceBegin and maximalExistenceEnd to the <validTime/> element of the <item/> element with specified itemTarget in the <logical/> container. |
| DeleteMaximalExistenceFromValidTime InItem(AD.xml,<br>itemTarget,<br>MaximalExistenceBegin,<br>MaximalExistenceEnd) | Removes the <maximalExistence/> element from the <validTime/> element of the <item/> element with specified itemTarget in the <logical/> container. |

| Primitive change | Description |
|---|---|
| SetFrequencyInValidTimeInItem(<br>AD.xml,<br>itemTarget,<br>validTimeFrequency) | Adds or changes the <frequency/> element with specified validTimeFrequency to the <validTime/> element of the <item/> element with specified itemTarget in the <logical/> container. |
| DeleteFrequencyFromValidTimeInItem(<br>AD.xml,<br>itemTarget) | Removes the <frequency/> element from the <validTime/> element of the <item/> element with specified itemTarget in the <logical/> container. |
| AddTransactionTimeToItem(AD.xml,<br>itemTarget,<br>transactionTimeKind,<br>transactionTimeContent,<br>transactionTimeExistence) | Adds the <transactionTime/> element with specified transactionTimeKind, transactionTimeContent, and transactionTimeExistence to the <item/> element with specified itemTarget in the <logical/> container, where the three last arguments are optional. |
| DeleteTransactionTimeFromItem(<br>AD.xml,<br>itemTarget) | Removes the <transactionTime/> element from the <item/> element with specified itemTarget in the <logical/> container. |
| SetTransactionTimeInItem(AD.xml,<br>itemTarget,<br>transactionTimeKind,<br>transactionTimeContent,<br>transactionTimeExistence) | Changes the value of the kind attribute and/or the value of the content attribute and/or the value of the existence attribute of the <transactionTime/> element of the <item/> element with specified itemTarget in the <logical/> container. |
| SetFrequencyInTransactionTimeInItem(<br>AD.xml,<br>itemTarget,<br>transactionTimeFrequency) | Adds or changes the <frequency/> element with specified transactionTimeFrequency to the <transactionTime/> element of the <item/> element with specified itemTarget in the <logical/> container. |
| DeleteFrequencyFromTransactionTimeIn<br>Item(AD.xml,<br>itemTarget) | Removes the <frequency/> element from the <transactionTime/> element of the <item/> element with specified itemTarget in the <logical/> container. |
| AddItemIdentifierToItem(AD.xml,<br>itemTarget,<br>itemIdentifierName,<br>itemIdentifierTimeDimension) | Adds the <itemIdentifier/> element with specified itemIdentifierName and itemIdentifierTimeDimension to the <item/> element with specified itemTarget in the <logical/> container. Possible values of the itemIdentifierTimeDimension argument: one of validTime, transactionTime, or bitemporal; default is validTime. |
| DeleteItemIdentifierFromItem(AD.xml,<br>itemTarget) | Removes the <itemIdentifier /> element from the <item/> element with specified itemTarget in the <logical/> container. |
| SetItemIdentifierInItem(AD.xml,<br>itemTarget,<br>itemIdentifierName,<br>itemIdentifierTimeDimension) | Changes the value of the name attribute and/or the value of the timeDimension attribute of the <item/> element with specified itemTarget in the <logical/> container. |
| AddKeyrefToItemIdentifier(AD.xml,<br>itemTarget,<br>keyrefName,keyrefType) | Adds the <keyref/> element with specified keyrefName and keyrefType to the<itemIdentifier/> element of the <item/> element with specified itemTarget in the <logical/> container. |
| DeleteKeyrefFromItemIdentifier(<br>AD.xml,<br>itemTarget,<br>keyrefName) | Removes the <keyref/> element with specified keyrefName from the<itemIdentifier/> element of the <item/> element with specified itemTarget in the <logical/> container. |
| ChangeKeyrefInItemIdentifier(AD.xml,<br>itemTarget,<br>oldkeyrefName,<br>newkeyrefName,<br>oldkeyrefType,<br>newkeyrefType) | Changes the value of the refName attribute and/or the value of the refType attribute of the <keyref/> element with specified oldkeyrefName in the <itemIdentifier/> element of the <item/> element with specified itemTarget in the <logical/> container. |
| AddFieldToItemIdentifier(AD.xml,<br>itemTarget,<br>fieldPath) | Adds the <field/> element with specified fieldPath to the <itemIdentifier/> element of the <item/> element with specified itemTarget in the <logical/> container. |
| DeleteFieldFromItemIdentifier(AD.xml,<br>itemTarget,<br>fieldPath) | Removes the <field/> element with specified fieldPath from the <itemIdentifier/> element of the <item/> element with specified itemTarget in the <logical/> container. |
| ChangeFieldInItemIdentifier(AD.xml,<br>itemTarget,<br>oldfieldPath,newfieldPath) | Changes the <field/> element with specified oldfieldPath in the <itemIdentifier/> element of the <item/> element with specified itemTarget in the <logical/> container. |
| AddAttributeToItem(AD.xml,<br>itemTarget,<br>attributeName) | Adds the <attribute/> element with specified attributeName to the <item/> element with specified itemTarget in the <logical/> container. |
| DeleteAttributeFromItem(AD.xml,<br>itemTarget,<br>attributeName) | Removes the <attribute/> element with specified attributeName from the <item/> element with specified itemTarget in the <logical/> container. |
| ChangeAttributeNameInItem(AD.xml,<br>itemTarget,<br>attributeName) | Changes the name attribute of the <attribute/> element in the <item/> element with specified itemTarget in the <logical/> container. |
| AddValidTimeToAttribute(AD.xml,<br>itemTarget,<br>attributeName, | Adds the <validTime/> element with specified validTimeKind and |

| Primitive change | Description |
|---|---|
| validTimeKind,<br>validTimeContent) | validTimeContent to the <attribute/> element with specified attributeName of the <item/> element with specified itemTarget in the <logical/> container, where validTimeKind is required and validTimeContent is optional.<br>Possible values of some arguments:<br>- validTimeKind: either state or event.<br>- validTimeContent: either constant or varying. |
| DeleteValidTimeFromAttribute(AD.xml,<br>itemTarget,<br>attributeName) | Removes the <validTime/> element from the <attribute/> element with specified attributeName of the <item/> element with specified itemTarget in the <logical/> container. |
| SetValidTimeInAttribute(AD.xml,<br>itemTarget,<br>attributeName,<br>validTimeKind,<br>validTimeContent) | Changes the value of the kind attribute and/or the value of the content attribute of the <validTime/> element of the <attribute/> element with specified attributeName of the <item/> element with specified itemTarget in the <logical/> container. |
| AddContentVaryingApplicabilityToValidTimeInAttribute(AD.xml,<br>itemTarget,<br>attributeName,<br>contentVaryingApplicabilityBegin,<br>contentVaryingApplicabilityEnd) | Adds the <contentVaryingApplicability/> element with specified contentVaryingApplicability-Begin and contentVaryingApplicability-End to the <validTime/> element of the <attribute/> element with specified attributeName in the <item/> element with specified itemTarget in the <logical/> container. |
| DeleteContentVaryingApplicabilityFromValidTimeInAttribute(<br>AD.xml,<br>itemTarget,<br>attributeName,<br>contentVaryingApplicabilityBegin,<br>contentVaryingApplicabilityEnd) | Removes the <contentVaryingApplicability/> element with specified oldcontentVaryingApplicabi-lityBegin and oldcontentVaryingApplicabi-lityEnd from the <validTime/> element of the <attribute/> element with specified attributeName in the <item/> element with specified itemTarget in the <logical/> container. |
| ChangeContentVaryingApplicabilityInValidTimeInAttribute(<br>AD.xml,<br>itemTarget,<br>attributeName,<br>oldcontentVaryingApplicabilityBegin,<br>oldcontentVaryingApplicabilityEnd,<br><br>newcontentVaryingApplicabilityBegin,<br>newcontentVaryingApplicabilityEnd) | Changes the value of the begin attribute and/or the value of the end attribute of the <contentVaryingApplicability/> element with specified oldcontentVaryingApplicabi-lityBegin and oldcontentVaryingApplicabi-lityEnd in the <validTime/> element of the <attribute/> element with specified attributeName in the <item/> element with specified itemTarget in the <logical/> container. |
| SetFrequencyInValidTimeInAttribute(<br>AD.xml,<br>itemTarget,<br>attributeName,<br>validTimeFrequency) | Adds or changes the <frequency/> element with specified validTimeFrequency to the <validTime/> element of the <attribute/> element with specified attributeName in the <item/> element with specified itemTarget in the <logical/> container. |
| DeleteFrequencyFromValidTimeInAttribute(AD.xml,<br>itemTarget,<br>attributeName) | Removes the <frequency/> element from the <validTime/> element of the <attribute/> element with specified attributeName in the <item/> element with specified itemTarget in the <logical/> container. |
| AddTransactionTimeToAttribute(<br>AD.xml,<br>itemTarget,<br>attributeName) | Adds the <transactionTime/> element to the <attribute/> element with specified attributeName in the <item/> element with specified itemTarget in the <logical/> container. |
| DeleteTransactionTimeFromAttribute(<br>AD.xml,<br>itemTarget,<br>attributeName) | Removes the <transactionTime/> element from the <attribute/> element with specified attributeName in the <item/> element with specified itemTarget in the <logical/> container. |
| SetFrequencyToTransactionTimeInAttribute(AD.xml,<br>itemTarget,<br>attributeName,<br>transactionTimeFrequency) | Adds or changes the <frequency/> element with specified transactionTimeFrequency to the <transactionTime/> element of the <attribute/> element with specified attributeName in the <item/> element with specified itemTarget in the <logical/> container. |
| DeleteFrequencyFromTransactionTimeInAttribute(AD.xml,<br>itemTarget,<br>attributeName) | Removes the <frequency/> element from the <transactionTime/> element of the <attribute/> element with specified attributeName in the <item/> element with specified itemTarget in the <logical/> container. |

For example, the effect of the AddItem(AD.xml, itemTarget) primitive, that is the contents of the AD.xml file after its application, is as follows:

AD.xml:

```
<?xml version="1.0" encoding="UTF-8"?>
<annotationSet
        xmlns="http://www.cs.arizona.edu/tau/tauXSchema/ASchema">
<physical>
  …
</physical>
<logical>
  <item target="itemTarget"/>
</logical>
</annotationSet>
```

The effect of the AddValidTimeToItem(AD.xml, itemTarget, validTimeKind, validTimeContent, validTimeExistence) primitive, that is the contents of the AD.xml file after its application, is as follows:

AD.xml:

```
<?xml version="1.0" encoding="UTF-8"?>
<annotationSet
        xmlns="http://www.cs.arizona.edu/tau/tauXSchema/ASchema">
<physical>
 …
</physical>
<logical>
 <item target="itemTarget">
  <validTime kind="validTimeKind" content="validTimeContent"
             existence="validTimeExistence"/>
 </item>
</logical>
</annotationSet>
```

The effect of the AddContentVaryingApplicabilityToValidTimeInItem(AD.xml, itemTarget, contentVaryingApplicabilityBegin, contentVaryingApplicabilityEnd) primitive, that is the contents of the AD.xml file after its application, is as follows:

AD.xml:

```
<?xml version="1.0" encoding="UTF-8"?>
<annotationSet
        xmlns="http://www.cs.arizona.edu/tau/tauXSchema/ASchema">
<physical>
 …
</physical>
<logical>
 <item target="itemTarget">
  <validTime kind="validTimeKind" content="validTimeContent"
             existence="validTimeExistence">
    <contentVaryingApplicability
                    begin="contentVaryingApplicabilityBegin"
                    end="contentVaryingApplicabilityEnd"/>
  </validTime>
 </item>
</logical>
</annotationSet>
```

Notice that it is possible to reduce the number of primitives if we consider some sub-elements which are common to the <item/> element and the <attribute/> element: the <validTime/> sub-element and its sub-elements (<contentVaryingApplicability/>, <maximalExistence/> and <frequency/>) and the <transactionTime/> sub-element and its sub-element (<frequency/>).

### *F. Illustrative Example*

Let us resume the example of the section 2.2. Suppose that on June 1, 2010, the designer decides to keep the history of the balance of each account along both transaction and valid times. Then, he/she changes the first version of the annotation document by modifying the item related to the Balance element: he/she adds an empty <validTime> element and changed the value of the timeDimension attribute of the <itemIdentifier> element from transactionTime to bitemporal. Suppose that he/she also decides to add another physical timestamp (having a bitemporal kind) to the element <Balance>. The second version of the annotation document is shown in Figure 5. Thus, the temporal schema is also updated by adding a new slice related to this new version of the annotation document, as shown in Figure 6.

The sequence of primitives that have been performed on the first version of the annotation document (BankAnnotations_V1.xml) to produce the second one (BankAnnotations_V2.xml) is as follows:

(i) AddValidTimeToItem("BankAnnotations_V1.xml", "/Bank/Account/Balance", state, varying, constant)

(ii) SetItemIdentifierInItem("BankAnnotations_V1.xml", "/Bank/Account/Balance", "balanceID", bitemporal)

(iii) AddStamp("BankAnnotations_V1.xml", "/Bank/Account/Balance", expandedVersion, bitemporal, extent)

```
<?xml version="1.0" encoding="UTF-8"?>
<annotationSet xmlns="http://www.cs.arizona.edu/tau/tauXSchema/
                                ASchema">
 <logical>
   <item target="/Bank/Account/Balance">
  <transactionTime kind="state" content="varying" existence="constant"/>
  <validTime kind="state" content="varying" existence="constant"/>
  <itemIdentifier name="balanceID" timeDimension="bitemporal">
   <field path="."/>
  </itemIdentifier>
   </item>
 </logical>
 <physical>
  <stamp target="/Bank/Account" dataInclusion="expandedVersion">
    <stampKind timeDimension="transactionTime" stampBounds="extent"/>
  </stamp>
  <stamp target="/Bank/Account/Balance"
            dataInclusion="expandedVersion">
    <stampKind timeDimension="bitemporal" stampBounds="extent"/>
  </stamp>
 </physical>
</annotationSet>
```

Figure 5. Second version of the annotation document (BankAnnotations_V2.xml), on June 1, 2010.

```
<?xml version="1.0" encoding="UTF-8"?>
<temporalSchema xmlns="http://www.cs.arizona.edu/tau/tauXSchema/
                                TSSchema">
   <conventionalSchema>
     <sliceSequence>
       <slice location="Bank_V1.xsd" begin="2010-02-01" />
     </sliceSequence>
   </conventionalSchema>
  <annotationSet>
     <sliceSequence>
       <slice location="BankAnnotations_V1.xml" begin="2010-02-01" />
       <slice location="BankAnnotations_V2.xml" begin="2010-06-01" />
     </sliceSequence>
  </annotationSet>
</temporalSchema>
```

Figure 6. Temporal schema (BankTemporalSchema.xml) on June 1, 2010.

On August 1, 2010, suppose that the designer decides to make the <Type> element varying in valid-time and to add a physical valid-time timestamp to this element. The third version of the annotation document is shown in Figure 7 and the updated temporal schema document is shown in Figure 8.

The sequence of primitives that have been performed on the second version of the annotation document (BankAnnotations_V2.xml) to produce the third one (BankAnnotations_V3.xml) is as follows:

(i) AddItem("BankAnnotations_V2.xml", "/Bank/Account/Type")

(ii) AddValidTimeToItem("BankAnnotations_V2.xml", "/Bank/Account/Type", state, varying, constant)

(iii) AddItemIdentifierToItem("BankAnnotations_V2.xml", "/Bank/Account/Type", "typeID", validTime)

(iv) AddFieldToItemIdentifier("BankAnnotations_V2.xml", "/Bank/Account/Type", ".")

(v) AddStamp("BankAnnotations_V2.xml", "/Bank/Account/Type", expandedVersion, validTime, extent)

## V. RELATED WORK

Schema versioning has been widely and deeply studied in the context of temporal relational databases (e.g. [5,13,14]) and temporal object-oriented databases (e.g. [15,16,17]). In the XML setting, a bibliography of work about temporal representation and evolution of documents and data on the web has been presented in [18].

In [19,20], the authors propose a generic approach for the management of schema versioning in multitemporal XML databases and for the manipulation of multitemporal data under schema versioning. This approach is based on the XML Schema language and is database-consistency preserving. However, this approach is low-level since operations are defined as modifications of XML elements/attributes.

In [21,22,23], the authors deal with the management of schema evolution and versioning in web information systems. However, in these proposals, the authors do not consider native XML databases (or repositories). They work always on a temporal relational database supporting schema evolution (or versioning) and they use XML to publish and to query data and schema. In particular, in [21], the authors propose SMO, a set of schema modification operators, to change the schema of relational databases. In [22], the authors address the issue of automatic SQL query rewriting after schema changes. In [23], the authors (i) propose ICOM, a set of integrity constraint modification operators, that completes SMO and allows designers to change some integrity constraints (as a part of the schema) in relational databases, and (ii) study rewriting of SQL updates after schema changes.

The papers, which are more strictly related with our work, are [7], [8] and [9]. In [7], the authors introduce τXSchema but did not discuss schema versioning. In [8] and [9], the authors deal with schema versioning in τXSchema but they focus only on capturing a time-varying schema and on validating documents against such a schema. [8] describes how the validator can be extended to validate temporal XML documents when their content (i.e., data) and also their schema are changing over time. [9] extends [8] by discussing how to accommodate gaps in the lifetime of an item, transaction semantics, and how to accommodate non-sequenced constraints across schema changes.

All previous works on τXSchema do not study how the schema changes are performed, or what schema change operations should be provided. The present paper extends previous work about τXSchema by proposing a complete and sound set of change primitives for physical and logical annotations and by defining their operational semantics.

```
<?xml version="1.0" encoding="UTF-8"?>
<annotationSet
        xmlns="http://www.cs.arizona.edu/tau/tauXSchema/ASchema">
 <logical>
  <item target="/Bank/Account/Balance">
   <transactionTime kind="state" content="varying" existence="constant"/>
   <validTime kind="state" content="varying" existence="constant"/>
   <itemIdentifier name="balanceID" timeDimension="bitemporal">
    <field path="."/>
   </itemIdentifier>
  </item>
  <item target="/Bank/Account/Type">
   <validTime kind="state" content="varying" existence="constant"/>
   <itemIdentifier name="typeID" timeDimension="validTime">
    <field path="."/>
   </itemIdentifier>
  </item>
 </logical>
 <physical>
  <stamp target="/Bank/Account" dataInclusion="expandedVersion">
   <stampKind timeDimension="transactionTime" stampBounds="extent"/>
  </stamp>
  <stamp target="/Bank/Account/Balance"
         dataInclusion="expandedVersion">
   <stampKind timeDimension="bitemporal" stampBounds="extent"/>
  </stamp>
  <stamp target="/Bank/Account/Type" dataInclusion="expandedVersion">
   <stampKind timeDimension="validTime" stampBounds="extent"/>
  </stamp>
 </physical>
</annotationSet>
```

Figure 7. Third version of the annotation document (BankAnnotations_V3.xml), on August 1, 2010.

```
<?xml version="1.0" encoding="UTF-8"?>
<temporalSchema
          xmlns="http://www.cs.arizona.edu/tau/tauXSchema/TSSchema">
  <conventionalSchema>
    <sliceSequence>
      <slice location="Bank_V1.xsd" begin="2010-02-01" />
    </sliceSequence>
  </conventionalSchema>
  <annotationSet>
    <sliceSequence>
      <slice location="BankAnnotations_V1.xml" begin="2010-02-01" />
      <slice location="BankAnnotations_V2.xml" begin="2010-06-01" />
      <slice location="BankAnnotations_V3.xml" begin="2010-08-01" />
    </sliceSequence>
  </annotationSet>
</temporalSchema>
```

Figure 8. Temporal schema (BankTemporalSchema.xml) on August 1, 2010.

## VI. CONCLUSION

This paper proposes a set of operations for the management of schema versioning in the τXSchema framework. In particular, a sound and complete set of schema change primitives for the maintenance of logical and physical annotations is introduced, and the syntax and operational semantics of each primitive are defined.

Currently, we are extending the present work by defining a set of schema change primitives for the conventional schema (i.e., change primitives acting on a standard XSD file). To this purpose, a completion in the τXSchema framework of the work started in [19] will be done.

In the future, we also plan to address temporal data

querying in the presence of multiple schema versions [14] in the τXSchema framework. The starting point for this extension will be the τXQuery language [24], which allows querying of τXSchema temporal documents under a single schema version.

## REFERENCES


[1] Extensible Markup Language (XML) 1.0 (Fifth Edition), W3C Recommendation, 26 November 2008. <http://www.w3.org/TR/2008/REC-xml-20081126/>.

[2] C. E. Dyreson and F. Grandi, "Temporal XML," in L. Liu and M. T. Özsu (Eds.), Encyclopedia of Database Systems. Heidelberg, Germany: Springer-Verlag, 2009, pp. 3032–3035.

[3] C. Zaniolo and F. Wang, "Temporal queries and version management in XML-based document archives," Data and Knowledge Engineering, vol. 65, May 2008, pp. 304-324.

[4] J. Clifford, A. Croker, F. Grandi, and A. Tuzhilin, "On Temporal Grouping," Proceedings of the International Workshop on Temporal Databases, Zürich, Switzerland, 17-18 September 1995, pp. 194–213.

[5] C. De Castro, F. Grandi, and M. R. Scalas, "Schema versioning for multitemporal relational databases," Information Systems, vol. 22, July 1997, pp. 249-290.

[6] J. F. Roddick, "Schema Versioning," in L. Liu and M. T. Özsu (Eds.), Encyclopedia of Database Systems. Heidelberg, Germany: Springer-Verlag, 2009, pp. 2499–2502.

[7] F. Currim, S. Currim, C. E. Dyreson, and R. T. Snodgrass, "A Tale of Two Schemas: Creating a Temporal XML Schema from a Snapshot Schema with τXSchema," Proceedings of the 9th International Conference on Extending Database Technology (EDBT 2004), Crete, Greece, 14-18 March 2004, pp. 348-365.

[8] C. E. Dyreson, R. T. Snodgrass, F. Currim, S. Currim, and S. Joshi, "Validating Quicksand: Schema Versioning in τXSchema," Proceedings of the 22nd International Conference on Data Engineering Workshops (ICDE Workshops 2006), Atlanta, GA, USA, 3-7 April 2006, pp. 82.

[9] C. E. Dyreson, R. T. Snodgrass, F. Currim, S. Currim, and Joshi S., "Validating Quicksand: Schema Versioning in τXSchema," Data Knowledge and Engineering, vol. 65, May 2008, pp. 223-242.

[10] XML Schema Part 0: Primer Second Edition, W3C Recommendation, 28 October 2004. <http://www.w3.org/TR/2004/REC-xmlschema-0-20041028/>

[11] F. Currim, S. Currim, C. E. Dyreson, S. Joshi, R. T. Snodgrass, S. W. Thomas, and E. Roeder, "τXSchema: Support for Data- and Schema-Versioned XML Documents," TimeCenter Technical Report TR-91, 279 pages, September 2009. <http://timecenter.cs.aau.dk/TimeCenterPublications/TR-91.pdf>

[12] Z. Brahmia, R. Bouaziz, F. Grandi, and B. Oliboni, "Schema Versioning in τXSchema-Based Multitemporal XML Repositories," TimeCenter Technical Report TR-93, 25 pages, December 2010. <http://timecenter.cs.aau.dk/TimeCenterPublications/TR-93.pdf>

[13] H.-C. Wei and R. Elmasri, "Schema versioning and database conversion techniques for bi-temporal databases," Annals of Mathematics and Artificial Intelligence, vol. 30, June 2000, pp. 23-52.

[14] F. Grandi, "A relational multi-schema data model and query language for full support of schema versioning," Proceedings of SEBD 2002 – National Conference on Advanced Database Systems, Isola d'Elba, Italy, 19-21 June 2002, pp. 323-336.

[15] L. Rodríguez, H. Ogata, and Y. Yano, "A temporal versioned object-oriented data schema model," Computers and Mathematics with Applications, vol. 41, January 2001, pp. 177–192.

[16] F. Grandi and F. Mandreoli, "A formal model for temporal schema versioning in object-oriented databases," Data and Knowledge Engineering, vol. 46, August 2003, pp. 123–167.

[17] R. M. Galante, C. S. Dos Santos, N. Edelweiss, and A. F. Moreira, "Temporal and versioning model for schema evolution in object-oriented databases," Data and Knowledge Engineering, vol. 53, May 2005, pp. 99–128.

[18] F. Grandi, "Introducing an Annotated Bibliography on Temporal and Evolution Aspects in the World Wide Web," ACM SIGMOD Record, vol. 33, June 2004, pp. 84-86.

[19] Z. Brahmia and R. Bouaziz, "An approach for schema versioning in multi-temporal XML databases," Proceedings of the 10th International Conference on Enterprise Information Systems (ICEIS 2008), Barcelona, Spain, 13-16 June 2008, Volume DISI, pp. 290-297.

[20] Z. Brahmia and R. Bouaziz, "Data Manipulation in Multi-Temporal XML Databases Supporting Schema Versioning," Proceedings of the 4th International EDBT Workshop on Database Technologies for Handling XML Information on the Web (DaTaX'09), Saint-Petersburg, Russia, 22 March 2009. <http://www.edbt.org/Proceedings/2009-StPetersburg/workshops/DataX09/papers/paper14.html>

[21] C. A. Curino, H. J. Moon, L. Tanca, and C. Zaniolo, "Schema Evolution in Wikipedia: toward a Web Information System Benchmark," Proceedings of the 10th International Conference on Enterprise Information Systems (ICEIS 2008), Barcelona, Spain, 13-16 June 2008, Volume DISI, pp. 323-332.

[22] C. A. Curino, H. J. Moon, and C. Zaniolo, "Graceful database schema evolution: the prism workbench," Proceedings of the 34th International Conference on Very Large Data Bases (VLDB 2008), Auckland, New Zealand, 24-30 August 2008, pp. 761-772.

[23] C. A. Curino, H. J. Moon, A. Deutsch, and C. Zaniolo, "Update Rewriting and Integrity Constraint Maintenance in a Schema Evolution Support System: PRISM++," Proceedings of the VLDB Endowment (PVLDB), vol. 4, November 2010, pp. 117-128.

[24] D. Gao and R. T. Snodgrass, "Temporal slicing in the evaluation of XML documents," Proceedings of the 29th International Conference on Very Large Data Bases (VLDB 2003), Berlin, Germany, 9-12 September 2003, pp. 632-643.