A Tale of Two Schemas: Creating a Temporal XML Schema from a Snapshot Schema with τXSchema

Faiz Currim[1], Sabah Currim[1], Curtis Dyreson[2], and Richard T. Snodgrass[1]

[1] University of Arizona, Tucson, AZ, USA
{fcurrim,scurrim}@bpa.arizona.edu, rts@cs.arizona.edu
[2] Washington State University, Pullman, WA, USA
cdyreson@wsu.edu

Abstract. The W3C XML Schema recommendation defines the structure and data types for XML documents. XML Schema lacks explicit support for time-varying XML documents. Users have to resort to ad hoc, non-standard mechanisms to create schemas for time-varying XML documents. This paper presents a data model and architecture, called τXSchema, for creating a temporal schema from a non-temporal (snapshot) schema, a temporal annotation, and a physical annotation. The annotations specify which portion(s) of an XML document can vary over time, how the document can change, and where timestamps should be placed. The advantage of using annotations to denote the time-varying aspects is that logical and physical data independence for temporal schemas can be achieved while remaining fully compatible with both existing XML Schema documents and the XML Schema recommendation.

1 Introduction

XML is becoming an increasingly popular language for documents and data. XML can be approached from two quite separate orientations: a *document-centered* orientation (e.g., HTML) and a *data-centered* orientation (e.g., relational and object-oriented databases). *Schemas* are important in both orientations. A schema defines the building blocks of an XML document, such as the types of elements and attributes. An XML document can be *validated* against a schema to ensure that the document conforms to the formatting rules for an XML document (is well-formed) and to the types, elements, and attributes defined in the schema (is valid). A schema also serves as a valuable guide for querying and updating an XML document or database. For instance, to correctly construct a query, e.g., in XQuery, a user will (usually) consult the schema rather than the data. Finally, a schema can be helpful in query optimization, e.g., in constructing a path index [24].

Several schema languages have been proposed for XML [22]. From among these languages, XML Schema is the most widely used. The syntax and semantics of XML Schema 1.0 are W3C recommendations [35, 36].

Time-varying data naturally arises in both document-centered and data-centered orientations. Consider the following wide-ranging scenarios. In a university, students take various courses in different semesters. At a company, job positions and salaries change. At a warehouse, inventories evolve as deliveries are made and good are

© Springer-Verlag Berlin Heidelberg 2004

shipped. In a hospital, drug treatment regimes are adjusted. And finally at a bank, account balances are in flux. In each scenario, querying the current state is important, e.g., "how much is in my account right now", but it also often useful to know how the data has changed over time, e.g., "when has my account been below $200".

An obvious approach would have been to propose changes to XML Schema to accommodate time-varying data. Indeed, that has been the approach taken by many researchers for the relational and object-oriented models [25, 29, 32]. As we will discuss in detail, that approach inherently introduces difficulties with respect to document validation, data independence, tool support, and standardization. So in this paper we advocate a novel approach that retains the non-temporal XML schema for the document, utilizing a series of separate schema documents to achieve data independence, enable full document validation, and enable improved tool support, while not requiring any changes to the XML Schema standard (nor subsequent extensions of that standard; XML Schema 1.1 is in development).

The primary contribution of this paper is to introduce the *τXSchema* (Temporal XML Schema) data model and architecture. τXSchema is a system for constructing schemas for time-varying XML documents[1]. A time-varying document records the evolution of a document over time, i.e., all of the versions of the document. τXSchema has a three-level architecture for specifying a schema for time-varying data[2]. The first level is the schema for an individual version, called the *snapshot schema*. The snapshot schema is a conventional XML Schema document. The second level is the *temporal annotations* of the snapshot schema. The temporal annotations identify which elements can vary over time. For those elements, the temporal annotations also effect a temporal semantics to the various integrity constraints (such as uniqueness) specified in the snapshot schema. The third level is the *physical annotations*. The physical annotations describe how the time-varying aspects are represented. Each annotation can be independently changed, so the architecture has (logical and physical) *data independence* [7]. Data independence allows XML documents using one representation to be automatically converted to a different representation while preserving the semantics of the data. τXSchema has a suite of auxiliary tools to manage time-varying documents and schemas. There are tools to convert a time-varying document from one physical representation to a different representation, to extract a time slice from that document (yielding a conventional static XML document), and to create a time-varying document from a sequence of static documents, in whatever representation the user specifies.

As mentioned, τXSchema *reuses* rather than extends XML Schema. τXSchema is consistent and compatible with both XML Schema and the XML data model. In τXSchema, a temporal validator augments a conventional validator to more comprehensively check the validity constraints of a document, especially temporal constraints that cannot be checked by a conventional XML Schema validator. We describe a means of validating temporal documents that ensures the desirable property of *snapshot validation subsumption*. We show elsewhere how a temporal document can be smaller and faster to validate than the associated XML snapshots [12].

[1] We embrace both the document and data centric orientations of XML and will use the terms "document" and "database" interchangeably.

[2] Three-level architectures are a common architecture in both databases [33] and spatio-temporal conceptual modeling [21].

While this paper concerns temporal XML Schema, we feel that the general approach of separate temporal and physical annotations is applicable to other data models, such as UML [28]. The contribution of this paper is two-fold: (1) introducing a three-level approach for logical data models and (2) showing in detail how this approach works for XML Schema in particular, specifically concerning a theoretical definition of snapshot validation subsumption for XML, validation of time-varying XML documents, and implications for tools operating on realistic XML schemas and data, thereby exemplifying in a substantial way the approach. While we are confident that the approach could be applied to other data models, designing the annotation specifications, considering the specifics of data integrity constraint checking, and ascertaining the impact on particular tools remain challenging (and interesting) tasks.

τXSchema focuses on *instance versioning* (representing a time-varying XML instance document) and not *schema versioning* [15, 31]. The schema can describe which aspects of an instance document change over time. But we assume that the schema itself is fixed, with no element types, data types, or attributes being added to or removed from the schema over time. *Intensional XML data* (also termed dynamic XML documents [1]), that is, parts of XML documents that consist of programs that generate data [26], are gaining popularity. Incorporating intensional XML data is beyond the scope of this paper.

The next section motivates the need for a new approach. Section 0 provides a theoretical framework for τXSchema, while an overview of its architecture is in Section 0. Details of the `τValidator` may be found in Section 0. Related work is reviewed in Section 0. We end with a summary and list of future work in Section 0.

2 Motivation

This section discusses whether conventional XML Schema is appropriate and satisfactory for time-varying data. We first present an example that illustrates how a time-varying document differs from a conventional XML document. We then pinpoint some of the limitations of XML Schema. Finally we state the desiderata for schemas for time-varying documents.

2.1 Motivating Example

Assume that the history of the Winter Olympic games is described in an XML document called `winter.xml`. The document has information about the athletes that participate, the events in which they participate, and the medals that are awarded. Over time the document is edited to add information about each new Winter Olympics and to revise incorrect information. Assume that information about the athletes participating in the 2002 Winter Olympics in Salt Lake City, USA was added on 2002-01-01. On 2002-03-01 the document was further edited to record the medal winners. Finally, a small correction was made on 2002-07-01.

To depict some of the changes to the XML in the document, we focus on information about the Norwegian skier Kjetil Andre Aamodt. On 2002-01-01 it was known that Kjetil would participate in the games and the information shown in Fig. 1

was added to `winter.xml`. Kjetil won a medal; so on 2002-03-01 the fragment was revised to that shown in Fig. 2. The edit on 2002-03-01 incorrectly recorded that Kjetil won a silver medal in the Men's Combined; Kjetil won a gold medal. Fig. 3 shows the correct medal information.

```
...
<athlete>
  <athName>Kjetil Andre Aamodt</athName>
</athlete>
...
```

Fig. 1. A fragment of `winter.xml` on 2002-01-01

```
<athlete>
  <athName>Kjetil Andre Aamodt</athName> won a medal in
  <medal mtype="silver">Men's Combined</medal>
</athlete>
```

Fig. 2. Kjetil won a medal, as of 2002-03-01

```
<athlete>
  <athName>Kjetil Andre Aamodt</athName> won a medal in
  <medal mtype="gold">Men's Combined</medal>
</athlete>
```

Fig. 3. Medal data is corrected on 2002-07-01

A *time-varying document* records a *version history*, which consists of the information in each version, along with timestamps indicating the lifetime of that version. Fig. 4 shows a fragment of a time-varying document that captures the history of Kjetil. The fragment is *compact* in the sense that each edit results in only a small, localized change to the document. The history is also *bi-temporal* because both the *valid time* and *transaction time* lifetimes are captured [20]. The *valid time* refers to the time(s) when a particular fact is true in the modeled reality, while the *transaction time* is the time when the information was edited. The two concepts are orthogonal. Time-varying documents can have each kind of time. In Fig. 4 the valid- and transaction-time lifetimes of each element are represented with an optional `<rs:timestamp>` sub-element[3]. If the timestamp is missing, the element has the same lifetime as its enclosing element. For example, there are two `<athlete>` elements with different lifetimes since the content of the element changes. The last version of `<athlete>` has two `<medal>` elements because the medal information is revised. There are many different ways to represent the versions in a time-varying document; the methods differ in which elements are timestamped, how the elements are timestamped, and how changes are represented (e.g., perhaps only differences between versions are represented).

Keeping the history in a document or data collection is useful because it provides the ability to recover past versions, track changes over time, and evaluate temporal queries [17]. But it changes the nature of validating against a schema. Assume that the

[3] The introduced `<rs:timestamp>` element is in the "`rs`" namespace to distinguish it from any `<timestamp>` elements already in the document. This namespace will be discussed in more detail in Sections 0 and 0.

```
...
<athlete>
  <rs:timestamp ttStart="2002-01-01" ttStop="2002-02-28"
    vtBegin="2002-02-01" vtEnd="2002-02-28"/>
  <athName>Kjetil Andre Aamodt</athName>
  ...
</athlete>
<athlete>
  <rs:timestamp ttStart="2002-03-01" ttStop="now"
    vtBegin="2002-03-01" vtEnd="now"/>
  <athName>Kjetil Andre Aamodt</athName> won a medal in
  <medal mtype="silver">
   <rs:timestamp ttStart="2002-03-01" ttStop="2002-06-30"
     vtAt="2002-03-01"/>
    Men's Combined
  </medal>
  <medal mtype="gold">
    <rs:timestamp ttStart="2002-07-01" ttStop="now" vtAt="2002-03-01"/>
    Men's Combined
  </medal>
...
```

Fig. 4. A fragment of a time-varying document

```
<element name="athlete">
  <complexType mixed="true">
    <sequence>
      <element name="athName" type="string"/>
      <element ref="medal" minOccurs="0" maxOccurs="unbounded"/>
      <element name="birthPlace" type="string" minOccurs="1"
        maxOccurs="1"/>
      <element name="phone" type="phoneNumType" minOccurs="0"
        maxOccurs="unbounded"/>
    </sequence>
    <attribute name="age" type="nonNegativeInteger" use="required"/>
  </complexType>
</element>
```

Fig. 5. An extract from the winOlympic schema

file `winOlympic.xsd` contains the *snapshot schema* for `winter.xml`. The snapshot schema is the schema for an individual version. The snapshot schema is a valuable guide for editing and querying individual versions. A fragment of the schema is given in Fig. 5. Note that the schema describes the structure of the fragment shown in Fig. 1, Fig. 2, and Fig. 3. The problem is that although individual versions conform to the schema, the time-varying document does not. So `winOlympic.xsd` cannot be used (directly) to validate the time-varying document of Fig. 4.

The snapshot schema could be used *indirectly* for validation by individually reconstituting and validating each version. But validating every version can be expensive if the changes are frequent or the document is large (e.g., if the document is a database). While the Winter Olympics document may not change often, contrast this with, e.g., a Customer Relationship Management database for a large company. Thousands of calls and service interactions may be recorded each day. This would lead to a very large number of versions, making it expensive to instantiate and validate each individually. The number of versions is further increased because there can be both valid time and transaction time versions.

To validate a time-varying document, a new, different schema is needed. The schema for a time-varying document should take into account the elements (and attributes) and their associated timestamps, specify the kind(s) of time involved, provide hints on how the elements vary over time, and accommodate differences in version and timestamp representation. Since this schema will express how the time-varying information is *represented*, we will call it the *representational schema*. The representational schema will be related to the underlying snapshot schema (Fig. 5), and allows the time-varying document to be validated using a conventional XML Schema validator (though not fully, as discussed in the next section).

2.2 Moving beyond XML Schema

Both the snapshot and representational schemas are needed for a time-varying document. The snapshot schema is useful in queries and updates. For example, a current query applies to the version valid now, a current update modifies the data in the current version, creating a new version, and a timeslice query extracts a previous version. All of these apply to a single version of a time-varying document, a version described by the snapshot schema. The representational schema is essential for validation and representation (storage). Many versions are combined into a single temporal document, described by the representational schema.

Unfortunately the XML Schema validator is *incapable* of fully validating a time-varying document using the representational schema. First, XML Schema is not sufficiently expressive to enforce *temporal constraints*. For example, XML Schema cannot specify the following (desirable) schema constraint: the transaction-time lifetime of a `<medal>` element should always be contained in the transaction-time lifetime of its parent `<athlete>` element. Second, a conventional XML Schema document augmented with timestamps to denote time-varying data cannot, in general, be used to validate a snapshot of a time-varying document. A snapshot is an instance of a time-varying document at a single point in time. For instance, if the schema asserts that an element is mandatory (`minOccurs=1`) in the context of another element, there is no way to ensure that the element is in every snapshot since the element's timestamp may indicate that it has a shorter lifetime than its parent (resulting in times during which the element is not there, violating this integrity constraint); XML Schema provides no mechanism for reasoning about the timestamps.

Even though the representational and snapshot schemas are closely related, there are no existing techniques to automatically derive a representational schema from a snapshot schema (or vice-versa). The lack of an automatic technique means that users have to resort to ad hoc methods to construct a representational schema. Relying on ad hoc methods limits data independence. The designer of a schema for time-varying data has to make a variety of decisions, such as whether to timestamp with periods or with *temporal elements* [16], which are sets of non-overlapping periods and which elements should be time-varying. By adopting a tiered approach, where the snapshot XML Schema, temporal annotations, and physical annotations are separate documents, individual schema design decisions can be specified and changed, often without impacting the other design decisions, or indeed, the processing of tools. For example, a tool that computes a snapshot should be concerned primarily with the snapshot schema; the logical and physical aspects of time-varying information should only affect (perhaps) the efficiency of that tool, not its correctness. With physical data independence, few applications that are unconcerned with representational details would need to be changed.

Finally, improved tool support for representing and validating time-varying information is needed. Creating a time-varying XML document and representational schema for that document is potentially labor-intensive. Currently a user has to manually edit the time-varying document to insert timestamps indicating when versions of XML data are valid (for valid time) or are present in the document (for transaction time). The user also has to modify the snapshot schema to define the syntax and semantics of the timestamps. The entire process would be repeated if a new timestamp representation were desired. It would be better to have automated tools to create, maintain, and update time-varying documents when the representation of the timestamped elements changes.

2.3 Desiderata

In augmenting XML Schema to accommodate time-varying data, we had several goals in mind. At a minimum, the new approach would exhibit the following desirable features.

- Simplify the representation of time for the user.
- Support a three-level architecture to provide data independence, so that changes in the logical and physical level are isolated.
- Retain full upward compatibly with existing standards and not require any changes to these standards.
- Augment existing tools such as validating parsers for XML in such a way that those tools are also upward compatible. Ideally, any off-the-shelf validating parser (for XML Schema) can be used for (partial) validation.
- Support both valid time and transaction time.
- Accommodate a variety of physical representations for time-varying data.
- Support instance versioning.

Note that while ad hoc representational schemas may meet the last three desiderata, they certainly don't meet the first four. Other desirable features, outside the scope of this paper, include supporting schema versioning and accommodating temporal indeterminacy and granularity.

3 Theoretical Framework

This section sketches the process of constructing a schema for a time-varying document from a snapshot schema. The goal of the construction process is to create a schema that satisfies the *snapshot validation subsumption* property, which is described in detail below. In the relational data model, a schema defines the structure of each relation in a database. Each relation has a very simple structure: a relation is a list of attributes, with each attribute having a specified data type. The schema also includes integrity constraints, such as the specification of primary and foreign keys. In a similar manner, an XML Schema document defines the valid structure for an XML document. But an XML document has a far more complex structure than a relation. A document is a (deeply) nested collection of elements, with each element potentially having (text) content and attributes.

3.1 Snapshot Validation Subsumption

Let D^T be an XML document that contains *timestamped elements*. A timestamped element is an element that has an associated timestamp. (A *timestamped attribute* can be modeled as a special case of a timestamped element.) Logically, the timestamp is a collection of times (usually periods) chosen from one or more temporal dimensions (e.g., valid time, transaction time). Without loss of generality, we will restrict the discussion in this section to lifetimes that consist of a single period in one temporal dimension[4]. The timestamp records (part of) the lifetime of an element[5]. We will use the notation x^T to signify that element x has been timestamped. Let the lifetime of x^T be denoted as $lifetime(x^T)$. One constraint on the lifetime is that the lifetime of an element must be contained in the lifetime of each element that encloses it[6].

The *snapshot operation* extracts a complete *snapshot* of a time-varying document at a particular instant. Timestamps are *not* represented in the snapshot. A snapshot at time t replaces each timestamped element x^T with its non-timestamped copy x if t is in $lifetime(x^T)$ or with the empty string, otherwise. The *snapshot* operation is denoted as

$$snp(t, D^T) = D$$

where D is the snapshot at time t of the time-varying document D^T.

Let S^T be a representational schema for a time-varying document D^T. The *snapshot validation subsumption property* captures the idea that, at the very least, the representational schema must ensure that every snapshot of the document is valid with respect to the *snapshot schema*. Let $vldt(S,D)$ represent the validation *status* of document D with respect to schema S. The status is true if the document is *valid* but false otherwise. Validation also applies to time-varying documents, e.g., $vldt^T(S^T, D^T)$ is the validation status of D^T with respect to a representational schema, S^T, using a temporal validator.

Property [Snapshot Validation Subsumption]. Let S be an XML Schema document, D^T be a time-varying XML document, and S^T be a representational schema, also an

[4] The general case is that a timestamp is a collection of periods from multiple temporal dimensions (a multidimensional temporal element).

[5] Physically, there are myriad ways to represent a timestamp. It could be represented as an `<rs:timestamp>` subelement in the content of the timestamped element as is done in the fragment in Fig. 4. Or it could be a set of additional attributes in the timestamped element, or it could even be a `<rs:version>` element that wraps the timestamped element.

[6] Note that the lifetime captures only when an element appears in the context of the enclosing elements. The same element can appear in other contexts (enclosed by different elements) but clearly it has a different lifetime in those contexts.

XML Schema document. S^T is said to have *snapshot validation subsumption* with respect to S if

$$vldt^T(S^T, D^T) \Leftrightarrow \forall t[t \in lifetime(D^T) \Rightarrow vldt(S, snp(t, D^T))]$$

Intuitively, the property asserts that a *good* representational schema will validate only those time-varying documents for which every snapshot conforms to the snapshot schema. The subsumption property is depicted in the following correspondence diagram.

$$\begin{tikzcd}
D^T \arrow[r, "vldt^T(S^T, D^T)"] \arrow[d, "snp(t, D^T)"'] & v \arrow[d, "v \Rightarrow w"] \\
D \arrow[r, "vldt(S, D)"'] & w
\end{tikzcd}$$

Fig. 6. Snapshot validation subsumption

Details of the process for constructing a schema for a time-varying document that conforms to the snapshot validation subsumption property from a snapshot schema are available in a technical report by the authors [12].

4 Architecture

The architecture of τXSchema is illustrated in Fig. 7. This figure is central to our approach, so we describe it in detail and illustrate it with the example. We note that although the architecture has many components, only those components shaded gray in the figure are specific to an individual time-varying document and need to be supplied by a user. New time-varying schemas can be quickly and easily developed and deployed. We also note that the representational schema, instead of being the only schema in an ad hoc approach, is merely an artifact in our approach, with the snapshot schema, temporal annotations, and physical annotations being the crucial specifications to be created by the designer.

The designer annotates the snapshot schema with *temporal annotations* (box 6). The temporal annotations together with the snapshot schema form the *logical* schema. Fig. 8 provides an extract of the temporal annotations on the winOlympic schema. The temporal annotations specify a variety of characteristics such as 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. For example, `<athlete>` is described as a state element, indicating that the `<athlete>` will be valid over a period (continuous) of time rather than a single instant. Annotations can be nested, enabling the target to be relative to that of its parent, and inheriting as

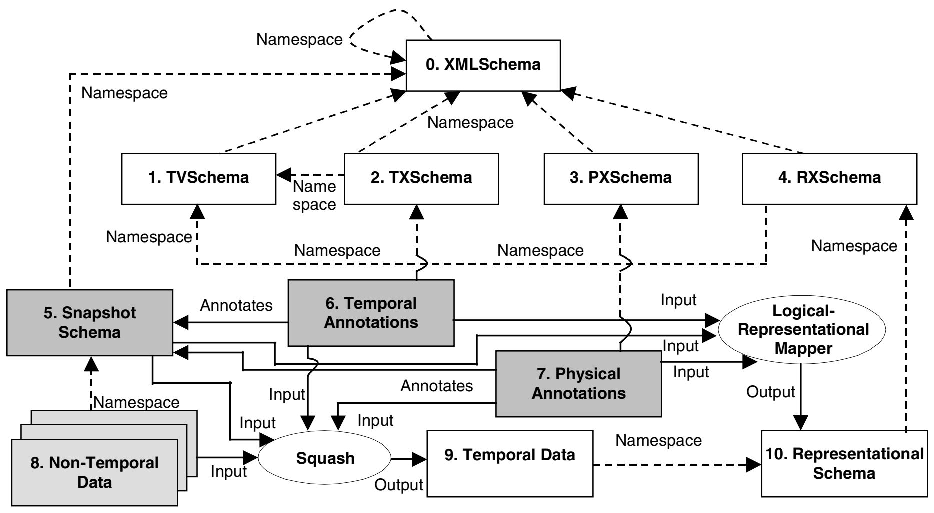

Fig. 7. Architecture of τXSchema

defaults the `kind`, `contentVarying`, and `existenceVarying` attribute values specified in the parent. The attribute `existenceVarying` indicates whether the element can be absent at some times and present at others. As an example, the presence of `existenceVarying` for an athlete's `phone` indicates that an athlete may have a phone at some points in time and not at other points in time. The attribute `contentVarying` indicates whether the element's content can change over time. An element's content is a string representation of its *immediate content*, i.e., text, sub-element names, and sub-element order.

Elements that are not described as time-varying are static and must have the same content and existence across every XML document in box 8. For example, we have assumed that the birthplace of an athlete will not change with time, so there is no annotation for `<birthPlace>` among the temporal annotations. The schema for the temporal annotations document is given by TXSchema (box 2), which in turn utilizes temporal values defined in a short XML Schema TVSchema (box 1). (Due to space limitations, we can't describe in detail these annotations, but it should be clear what aspects are specified here.)

The next design step is to create the *physical annotations* (box 7). In general, the physical annotations specify the timestamp representation options chosen by the user. An excerpt of the physical annotations for the winOlympic schema is given in Fig. 9. Physical annotations may also be nested, inheriting the specified attributes from their parent; these values can be overridden in the child element.

Physical annotations play two important roles.

1. They help to define where the physical timestamps will be placed (versioning level). The location of the timestamps is independent of which components vary over time (as specified by the temporal annotations). Two documents with the same logical information will look very different if we change the location of the physical timestamp. For example, although the elements `phone` and `athName` are time-varying, the user may choose to place the physical timestamp at the

```
<temporalAnnotations
  xmlns="http://www.cs.arizona.edu/tau/tauXSchema/TXSchema"
   xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
   xsi:schemaLocation="http://www.cs.arizona.edu/tau/tauXSchema/TXSchema.xsd">
<snapshotSchema schemaLocation="http://www.cs.arizona.edu/
   tau/tauXSchema/examples/schemas/winOlympic.xsd"/>
...
  <validTime target="/winOlympic/…/athlete" kind="state" contentVarying="true">
     <validTime target="@age"/>
     <validTime target="athName"/>
     <validTime target="medal" kind="event"/>
     <validTime target="phone" existenceVarying="true"/>
  </validTime>
...
  <transactionTime target="/winOlympic"/>
  <transactionTime target="/winOlympic/…/athlete/@age"/>
  <transactionTime target="/winOlympic/…/athlete/athName"/>
...
</temporalAnnotations>
```

Fig. 8. Sample temporal annotations

```
<physicalAnnotations xmlns= "http://www.cs.arizona.edu/tau/tauXSchema/PXSchema"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="http://www.cs.arizona.edu/tau/tauXSchema/PXSchema.xsd">

  <temporalAnnotations schemaLocation="http://www.cs.arizona.edu/
    tau/tauXSchema/examples/schemas/winOlympicTemporal.xml"/>
...
  <stampPosition target="/winOlympic" transactionTimeStampType="step" />
  <stampPosition target="/winOlympic/.../athlete" validTimeStampType="extent">
     <stampPosition target="@age" validTimeStampType="step"
       transactionTimeStampType="step"/>
     <stampPosition target="athName" transactionTimeStampType="step"/>
     <stampPosition target="medal" validTimeStampType="none" />
     <stampPosition target="phone" transactionTimeStampType="extent"/>
  </stampPosition>
...
</physicalAnnotations>
```

Fig. 9. Sample physical annotations

`athlete` level. Whenever any element below `athlete` changes, the entire `athlete` element is repeated.

2. The physical annotations also define the type of timestamp (for both valid time and transaction time). A timestamp can be one of two types: `step` or `extent`. An extent timestamp specifies both the start and end instants in the timestamp's period. In contrast a step-wise constant (`step`) timestamp represents only the start instant. The end instant is implicitly assumed to be just prior to the start of the next version, or *now* for the current version. However, one cannot use `step` timestamps when there might be "gaps" in time between successive versions. `Extent` timestamps do not have this limitation. Changing even one timestamp from `step` to `extent` can make a big difference in the representation.

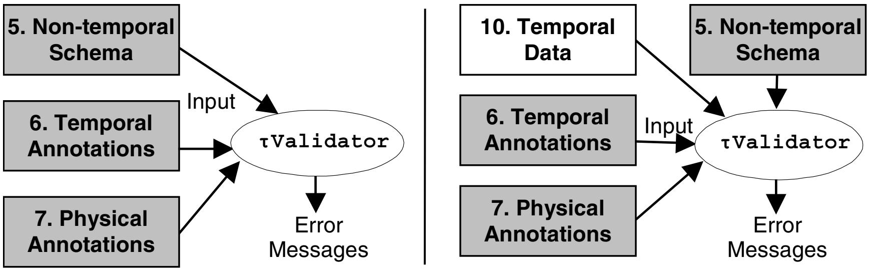

Fig. 10. τValidator: Checking the schemas and instance

The schema for the physical annotations document is PXSchema (box 3). τXSchema supplies a default set of physical annotations, which is to timestamp the root element with valid and transaction time using `step` timestamps, so the physical annotations are optional. (Again, space limitations do not allow us to describe these annotations in detail.)

We emphasize that our focus is on capturing relevant aspects of physical representations, not on the specific representations themselves, the design of which is itself challenging. Also, since the temporal and physical annotations are orthogonal and serve two separate goals, we choose to maintain them independently. A user can change where the timestamps are located, independently of specifying the temporal characteristics of that particular element. In the future, when software environments for managing changes to XML files over time are available, the user could specify temporal and physical annotations for an element together (by annotating a particular element to be temporal and also specifying that a timestamp should be located at that element), but these would remain two distinct aspects from a conceptual standpoint.

At this point, the designer is finished. She has written one conventional XML schema (box 5) and specified two sets of annotations (boxes 6 and 7). We provide boxes 1, 2, 3, and 4; XML Schema (box 0) is of course provided by W3C.

Let's now turn our attention to the tools that operate on these various specifications. The temporal annotations document (box 6) is passed through the `τValidator` (see the left half of Fig. 10) which checks to ensure that the annotations are consistent with the snapshot schema. The `Validator` utilizes the conventional validator for many of its checks. For instance, it validates the temporal annotations against the TXSchema. But it also checks that the temporal annotations are not inconsistent. Similarly, the physical annotations document is passed through the `τValidator` to ensure consistency of the physical annotations.

Once the annotations are found to be consistent, the *Logical to Representational Mapper* (software oval, Fig. 7) generates the *representational schema* (box 10) from the original snapshot schema and the temporal and physical annotations. The representational schema (mentioned in Section 0 as "`rs:`") is needed to serve as the schema for a time-varying document/data (box 9). The time-varying data can be created in four ways: 1) automatically from the non-temporal data (box 8) using τXSchema's `squash` tool (described in our technical report [12]), 2) automatically from the data stored in a database, i.e., as the result of a "temporal" query or view, 3) automatically from a third-party tool, or 4) manually.

The time-varying data is validated against the representational schema in two stages. First, a conventional XML Schema validating parser is used to parse and validate the time-varying data since the representational schema is an XML Schema document that satisfies the snapshot validation subsumption property. But as emphasized in Section 0, using a conventional XML Schema validating parser is not sufficient due to the limitations of XML Schema in checking temporal constraints. For example, a regular XML Schema validating parser has no way of checking something as basic as "the valid time boundaries of a parent element must encompass those of its child". These types of checks are implemented in the `τValidator`. So the second step is to pass the temporal data to `τValidator` as shown in the right half of Fig. 10. A temporal XML data file (box 9) is essentially a timestamped representation of a sequence of non-temporal XML data files (box 8). The namespace is set to its associated XML Schema document (i.e. representational schema). The timestamps are based on the characteristics defined in the temporal and physical annotations (boxes 6 and 7). The `τValidator`, by checking the temporal data, effectively checks the non-temporal constraints specified by the snapshot schema simultaneously on all the instances of the non-temporal data (box 8), as well as the constraints *between* snapshots, which cannot be expressed in a snapshot schema.

To reiterate, the conventional approach has the user start with a representational schema (box 10); our proposed approach is to have the user design a snapshot schema and two annotations, with the representational schema automatically generated.

5 Tools

Our three-level schema specification approach enables a suite of tools operating both on the schemas and the data they describe. The tools are open-source and beta versions are available[7]. The tools were implemented in Java using the DOM API [34]. We now turn to a central tool, the temporal validator.

The logical and physical temporal annotations (Fig. 7, boxes 6 and 7) for a non-temporal XML Schema (Fig. 7, box 5) are XML documents and hence can be validated as such. However, a validating XML parser cannot perform all of the necessary checks to ensure that the annotations are correctly specified. For example it cannot check that elements that have a `minOccurs` of 0 do not use a step-wise constant timestamp representation (i.e. a compact representation that assumes continuous existence, and where only the begin/start time of a timestamp is specified and the end/stop time of a timestamp is assumed to be the same as the begin/start point of the succeeding timestamp). This motivates the need for a special validator for the temporal and physical annotations. We implemented a tool, called `Validator`, to check the annotations. First, `τValidator` validates the temporal and physical annotations against the TXSchema and PXSchema, respectively. Then it performs additional tests to ensure that the snapshot schema and the temporal and physical annotations are all consistent.

[7] `http://www.cs.arizona.edu/tau/txschema/` and `http://www.cs.arizona.edu/tau/tdom/`

τ`Validator` also validates time-varying data. A temporal data validator must ensure that every snapshot of the time-varying document conforms to the snapshot schema. It does this, in part, by using an existing XML Schema validating parser to validate the temporal document against the representational schema. `Validator` then performs two additional kinds of checks: representational checks and checks to compensate for differences in the semantics of temporal and non-temporal constraints. For instance it needs to check that there are no gaps in the lifetimes of versions for elements that have `minOccurs=1` in the representational schema.

Additional details about other tools developed—including results of experiments performed on the tools—are available elsewhere [12].

6 Review of Related Work

While there have been a number of research efforts that have identified methods to detect and represent changes in XML documents over time [18], none have addressed the issue of validating a time-varying document.

There are various XML schemas that have been proposed in the literature and in the commercial arena. We chose to extend XML Schema in τXSchema because it is backed by the W3C and supports most major features available in other XML schemas [22]. It would be relatively straightforward to apply the concepts in this paper to develop time support for other XML schema languages; less straightforward but possible would be to apply our approach of temporal and physical annotations to other data models, such as UML [28].

Garcia-Molina and Cho [10] provide evidence that some web pages change on every access, whereas other pages change very infrequently, with a coarse average change interval of a web page of 4 months. Nguyen et al. [27] describe how to detect changes in XML documents that are accessible via the web. In the Xyleme system [37], the XML Alerter module periodically (with a periodicity specified by the user) accesses the XML document and compares it with a cached version of the document. The result is a sequence of static documents, each with an associated existence period. Dyreson [13] describes how a web server can capture some of the versions of a time-varying document, by caching the document as it is served to a client, and comparing the cached version against subsequent requests to see if anything has changed. Amagasa et al. [2] classify the methods used to access XML documents into two general categories: (i) using specialized APIs for XML documents, such as DOM, and (ii) directly editing documents, e.g., with an editor. In the former case, to access and modify temporal XML documents, DOM can be extended to automatically capture temporal information (and indeed, we have implemented such functionality in τDOM). It is also possible to capture transaction time information in the documents through change analysis, as discussed above and elsewhere [4, 11].

There has been a lot of interest in representing time-varying documents. Marian et al. [23] discuss versioning to track the history of downloaded documents. Chien, Tsotras and Zaniolo [9] have researched techniques for compactly storing multiple versions of an evolving XML document. Chawathe et al. [8] described a model for representing changes in semi-structured data and a language for querying over these changes. For example, the *diff* based approach [4, 11] focuses on an efficient way to store time-varying data and can be used to help detect transaction time changes in the document at the physical level. Buneman et al. [6] provide another means to store a single copy of an element that occurs in many snapshots. Grandi and Mandreoli [19] propose a `<valid>` tag to define a validity context that is used to timestamp part of a document. Finally, Chawathe et al. [8] and Dyreson et al. [14] discuss timestamps on edges in a semi-structured data model.

Recently there has been interest in incremental validation of XML documents [3, 5, 30]. These consider validating a snapshot that is the result of updates on the previous snapshot, which has already been validated. In a sense, this is the dual to the problem we consider, which is validating a (compressed) temporal document all at once, rather than once per snapshot (incrementally or otherwise).

None of the approaches above focus on the extensions required in XML Schema to adequately specify the nature of changes permissible in an XML document over time, and the corresponding validation of the extended schema. In fact, some of the previous approaches that attempt to identify or characterize changes in documents do not consider a schema. As our emphasis is on logical and physical data modeling, we assume that a schema is available from the start, and that the desire is for that schema to capture both the static and time-varying aspects of the document. If no schema exists, tools can derive the schema from the base documents, but that is beyond the scope of this paper. Our approach applies at the logical view of the data, while also being able to specify the physical representation. Since our approach is independent of the physical representation of the data, it is possible to incorporate the *diff*-based approach and other representational approaches [6] in our physical annotations.

7 Conclusion

In this paper we introduce the τXSchema model and notation to annotate XML Schemas to support temporal information. τXSchema provides an efficient way to annotate temporal elements and attributes. Our design conforms to W3C XML Schema definition and is built on top of XML Schema. Our approach ensures data independence by separating (i) the snapshot schema document for the instance document, (ii) information concerning which portion(s) of the instance document can vary over time, and (iii) where timestamps should be placed and precisely how the time-varying aspects should be represented. Since these three aspects are orthogonal, our approach allows each aspect to be changed independently. A small change to the physical annotations (or temporal annotations) can effect a large change in the (automatically generated) representational schema and the associated XML file.

This separation of concerns may be exploited in supporting tools; several new, quite useful tools are discussed that exploit the logical and physical data independence provided by our approach. Additionally, this independence enables existing tools (e.g., the XML Schema validator, XQuery, and DOM) to be used in the implementation of their temporal counterparts.

Future work includes extending the τXSchema model to fulfill the remaining issues in the desiderata and beyond. Indeterminacy and granularity are two significant and related issues, and should be fully supported by τXSchema. We anticipate that providing this support would require additions to the TVSchema / TXSchema / PXSchema / RXSchema (Fig. 7, boxes 1–4), but no changes to the user-designed schemas (Fig. 7, boxes 5–7). These augmentations would be upward compatibile with this version of τXSchema and be transparent to the user. Schema versioning is another important capability. For simplicity, we assume that the XML document changes, but the schema remains stable over time. However, in reality, the schema will also change with time. We are designing an extension that takes into account schema versioning.

We plan to extend our approach to also accommodate intensional XML data [26] which refer to programs that generate data. Some of these programs may be evaluated (a process termed *materialization*), with the results replacing the programs in the document. There are several interesting time-varying aspects of intensional XML data: (i) the programs themselves may change over time, (ii) even if the programs are static, the results of program evaluations may change over time, as external data the programs access changes, and (iii) even if the programs and the external data are static, different versions of the program evaluators (e.g., Java compiler) may be present, may generate different results due to incompatibilities between versions. It is challenging to manage this combination of schema and instance versioning over time.

Another broad area of work is optimization and efficiency. Currently there is no separation of elements or attributes based on the relative frequency of update. In the situation that some elements (for example) vary at a significantly different rate than other elements, it may prove more efficient to split the schema up into pieces such that elements with similar “rates of change” are together [25, 29, 32]. This would avoid redundant repetition of elements that do not change as frequently. Related to optimization is the issue of optimizing the use of time-varying text content. For instance it may be desirable to capture order among different pieces of text content within an element (e.g., different pieces may be used to describe a particular sub-element and may therefore vary with a frequency strongly correlated to the sub-element’s temporal characteristics). We want to incorporate recently proposed representations (e.g., [4, 6, 9, 11]) into our physical annotations. Finally, the efficiency of the tools mentioned in Section 5 can be improved. For example, it would be interesting to investigate whether incremental validation approaches [3, 5, 30] are applicable in the temporal schema validator.

References

[1] Abiteboul, S., Bonifati, A., Cobena, G., Manolescu, I. and Milo, T., Dynamic XML Documents with Distribution and Replication. *Proceedings of the ACM SIGMOD International Conference on Management of Data*, (San Diego, CA, 2003), 527-538.

[2] Amagasa, T., Yoshikawa, M. and Uemura, S., A Data Model for Temporal XML Documents. *Proceedings of the 11th International Workshop on Database and Expert Systems Applications*, (London, England, 2000), Springer, Berlin, New York, 334-344.

[3] Barbosa, D., Mendelzon, A., Libkin, L., Mignet, L. and Arenas, M., Efficient Incremental Validation of XML Documents. *Proceedings of the 20th International Conference on Data Engineering*, (Boston, MA, 2004), IEEE Computer Society.

[4] Birsan, D., Sluiman, H. and Fernz, S.-A. XML Diff and Merge Tool, IBM alphaWorks, 1999. http://www.alphaworks.ibm.com/tech/xmldiffmerge.

[5] Bouchou, B. and Halfeld-Ferrari, M., Updates and Incremental Validation of XML Documents. *Proceedings of the 9th International Workshop on Data Base Programming Languages*, (Potsdam, Germany, 2003), Springer.

[6] Buneman, P., Khanna, S., Tajima, K. and Tan, W.C., Archiving scientific data. *Proceedings of the ACM SIGMOD International Conference on Management of Data*, (Madison, WI, 2002), ACM, 1-12.

[7] Burns, T., Fong, E.N., Jefferson, D., Knox, R., Mark, L., Reedy, C., Reich, L., Roussopoulos, N. and Truszkowski, W. Reference Model for DBMS Standardization, Database Architecture Framework Task Group of the ANSI/X3/SPARC Database System Study Group. *SIGMOD Record*, *15* (1). 19-58, 1986.

[8] Chawathe, S., Abiteboul, S. and Widom, J., Representing and Querying Changes in Semistructured Data. *Proceedings of the 14th International Conference on Data Engineering*, (Orlando, FL, USA, 1998), IEEE Computer Society, 4-13.

[9] Chien, S., Tsotras, V. and Zaniolo, C. Efficient schemes for managing multiversion XML documents. *VLDB Journal*, *11* (4). 332-353, 2002.

[10] Cho, J. and Garcia-Molina, H., The Evolution of the Web and Implications for an Incremental Crawler. *Proceedings of the 26th International Conference on Very Large Data Bases*, (Cairo, Egypt, 2000), Morgan Kaufmann, 200-209.

[11] Cobena, G., Abiteboul, S. and Marian, A., Detecting Changes in XML Documents. *Proceedings of the 18th International Conference on Data Engineering*, (San Jose, California, 2002), IEEE Computer Society, 41-52.

[12] Currim, F., Currim, S., Snodgrass, R.T. and Dyreson, C.E. τXSchema: Managing Temporal XML Schemas, Technical Report TR-77, TimeCenter, 2003.

[13] Dyreson, C., Towards a Temporal World-Wide Web: A Transaction Time Web Server. *Proceedings of the 12th Australasian Database Conference*, (Gold Coast, Australia, 2001), 169-175.

[14] Dyreson, C.E., Bohlen, M. and Jensen, C.S., Capturing and Querying Multiple Aspects of Semistructured Data. *Proceedings of the 25th International Conference on Very Large Data Bases*, (Edinburgh, Scotland, UK, 1999), Morgan Kaufmann, 290-301.

[15] Franconi, E., Grandi, F. and Mandreoli, F., Schema Evolution and Versioning: A Logical and Computational Characterisation. *Database Schema Evolution and Meta-Modeling, Proceedings of the 9th International Workshop on Foundations of Models and Languages for Data and Objects, FoMLaDO/DEMM*, (Dagstuhl, Germany, 2000), Springer, 85-99.

[16] Gadia, S. A Homogeneous Relational Model and Query Languages for Temporal Databases. *ACM Transactions on Database Systems*, *13* (4). 418-448, 1988.

[17] Gao, D. and Snodgrass, R.T., Temporal Slicing in the Evaluation of XML Queries. *Proceedings of the 29th International Conference on Very Large Databases*, (Berlin, Germany, 2003), Morgan Kaufmann, 632-643.

[18] Grandi, F. An Annotated Bibliography on Temporal and Evolution Aspects in the WorldWideWeb, Technical Report TR-77, TimeCenter, 2003.

[19] Grandi, F. and Mandreoli, F. The Valid Web: its time to Go…, Technical Report TR-46, TimeCenter, 1999.

[20] Jensen, C.S. and Dyreson, C.E. (eds.). A Consensus Glossary of Temporal Database Concepts. in Etzion, O., Jajodia, S. and Sripada, S. (eds.). *Temporal Databases: Research and Practice*, Springer-Verlag, 1998, 367-405.

[21] Khatri, V., Ram, S. and Snodgrass, R.T. Augmenting a Conceptual Model with Spatio-Temporal Annotations. *IEEE Transactions on Knowledge and Data Engineering*, forthcoming, 2004.

[22] Lee, D. and Chu, W. Comparative Analysis of Six XML Schema Languages. *SIGMOD Record*, *29* (3). 76-87, 2000.

[23] Marian, A., Abiteboul, S., Cobena, G. and Mignet:, L., Change-Centric Management of Versions in an XML Warehouse. *Proceedings of the Very Large Data Bases Conference*, (Roma, Italy, 2001), Morgan Kaufmann, 581-590.

[24] McHugh, J. and Widom, J., Query Optimization for XML. *Proceedings of the 25th International Conference on Very Large Databases*, (Edinburgh, Scotland, UK, 1999), Morgan Kaufmann, 315-326.

[25] McKenzie, E. and Snodgrass, R.T. An Evaluation of Relational Algebras Incorporating the Time Dimension in Databases. *ACM Computing Surveys*, *23* (4). 501-543, 1991.
[26] Milo, T., Abiteboul, S., Amann, B., Benjelloun, O. and Ngoc, F.D., Exchanging Intensional XML Data. *Proceedings of the ACM SIGMOD International Conference on Management of Data*, (San Diego, CA, 2003), 289-300.
[27] Nguyen, B., Abiteboul, S., Cobena, G. and Preda, M., Monitoring XML Data on the Web. *Proceedings of the ACM SIGMOD International Conference on Management of Data*, (Santa Barbara, CA, 2001), 437-448.
[28] OMG. Unified Modeling Language (UML), v1.5, 2003. http://www.omg.org/technology/documents/formal/uml.htm.
[29] Ozsoyoglu, G. and Snodgrass, R.T. Temporal and Real-Time Databases:A Survey. *IEEE Transactions on Knowledge and Data Engineering*, *7* (4). 513-532, 1995.
[30] Papakonstantinou, Y. and Vianu, V., Incremental Validation of XML Documents. *Proceedings of the 9th International Conference on Database Theory*, (Siena, Italy, 2003), Springer, 47-63.
[31] Roddick, J.F. A Survey of Schema Versioning Issues for Database Systems. *Information and Software Technology*, *37* (7). 383-393, 1995.
[32] Snodgrass, R.T. Temporal Object-Oriented Databases: A Critical Comparison. in Kim, W. ed. *Modern Database Systems: The Object Model, Interoperability and Beyond*, Addison-Wesley/ACM Press, 1995, 386-408.
[33] Steel, T.B., Jr., Chairman Interim Report: ANSI/X3/SPARC Study Group on Data Base Management Systems 75-02-08. *FDT-Bulletin of ACM SIGMOD*, *7* (2). 1-140, 1975.
[34] W3C. Document Object Model (DOM) Level 2 HTML Specification Version 1.0. Hors, A.L. ed., W3C, 2002. http://www.w3.org/TR/2002/PR-DOM-Level-2-HTML-20021108/.
[35] W3C. XML Schema Part 1: Structures. Mendelsohn, N. ed., W3C, 2001. http://www.w3.org/TR/2001/REC-xmlschema-1-20010502/.
[36] W3C. XML Schema Part 2: Datatypes. Malhotra, A. ed., W3C, 2001. http://www.w3.org/TR/2001/REC-xmlschema-2-20010502/.
[37] Xyleme, L. A dynamic warehouse for XML Data of the Web. *IEEE Data Engineering Bulletin*, *24* (2). 40-47, 2001.