Weaving Temporal and Reliability Aspects into a Schema Tapestry

Curtis Dyreson[1], Richard T. Snodgrass[2], Faiz Currim[3], Sabah Currim[4], and Shailesh Joshi[5]

[1] Washington State University, Pullman, WA, cdyreson@eecs.wsu.edu
[2] University of Arizona, Tucson, AZ, rts@cs.arizona.edu
[3] University of Iowa, Iowa City, IA, faiz-currim@uiowa.edu
[4] Florida State University, Tallahassee, FL, scurrim@ci.fsu.edu
[5] University of Arizona, Tucson, AZ, shaileshpjoshi@gmail.com

Abstract. In aspect-oriented programming (AOP) a cross-cutting concern is implemented in an *aspect*. An aspect *weaver* blends code from the aspect into a program's code at programmer-specified *cut points*, yielding an aspect-enhanced program. In this paper we apply some of the concepts from the AOP paradigm to data. Like code, data also has cross-cutting concerns such as versioning, security, privacy, and reliability. We propose modeling a cross-cutting data concern as a *schema aspect*. A schema aspect describes the structure of the metadata in the cross-cutting concern, identifies the types of data elements that can be wrapped with metadata, i.e., the cut points, and provides some simple constraints on the use of the metadata. Several schema aspects can be applied to a single data collection, though in this paper we focus on just two aspects: a reliability aspect and a temporal aspect. We show how to weave the schema for these two aspects together with the schema for the data into a single, unified schema that we call a *schema tapestry*. The tapestry guides the construction, interpretation, and validation of an aspect-enhanced data collection.

1 Introduction

Aspect-oriented programming (AOP) is a new programming paradigm [19]. AOP arose from the need to quickly and safely add *cross-cutting concerns*, such as the monitoring of memory use or event logging, to a program *without manually modifying the program*. In AOP, a cross-cutting concern is implemented in an *aspect*. An aspect *weaver* blends code from the aspect into a program's code at programmer-specified *cut points*, yielding an aspect-enhanced program. A key benefit of AOP is that an aspect can be specified and implemented once, yet woven into many separate programs.

The AOP paradigm is *not directly applicable* to data since the goal of AOP is to modify *dynamic* program behavior, while data is *static*. But data, like code, also has cross-cutting concerns. One example is *versioning*. Many temporal and object-oriented data models have been developed to support the versioning of individual data items (c.f., [22]). Generally, these models anno-

tate the data with timestamps that signify the lifetime of each item. Though the timestamps are embedded in the data, the timestamps are actually *metadata*, that is, they are "data about data". Temporal data models have special behavior for handling timestamps embedded in data, for instance, to ensure that the timestamps are maintained during update and consulted during query processing.

Though not directly applicable, the AOP paradigm can be adapted to address cross-cutting concerns in schema design. Many cross-cutting concerns impinge on a schema. Data can be *annotated* with descriptions of where it came from, who inserted or changed it, and what its quality is [4][27]. The *provenance* of the data, what manipulations were performed on it to get it to this point, can also be recorded [6][7]. Similarly, the *accuracy* and *lineage* of the data can be captured [5][41]. *Security* and *privacy* introduce additional needs for metadata about particular data, such as who has access and to whom has information been released. *Reliability* and *performance* requirements are also cross-cutting concerns.

Though many schemas already exist for data, few, if any, model cross-cutting concerns. AOP can be adapted to schema design by implementing a cross-cutting data concern in an aspect, one aspect per kind of metadata, e.g., a temporal aspect for temporal metadata or a reliability aspect that captures metadata related to completeness and accuracy. The aspect is primarily a description of the static properties of the metadata, that is, the schema of the metadata and some simple constraints on its use. A *schema weaver* blends annotations from the aspect into the data's schema at designer-specified cut points, yielding an aspect-enhanced schema, which we call a *schema tapestry*. The schema tapestry is used as a guide in validating, interpreting, editing and querying data with embedded metadata. A key benefit of our approach is that a schema aspect can be designed once, yet woven into many separate data schemas.

The medium of our research is the Extensible Markup Language (XML). XML is fast becoming an important language for publishing and exchanging data on the web. XML is popular, in part, because data formatted in XML can be automatically processed to extract items of interest, e.g., using DOM or XQuery. An XML schema describes the structure of XML data. The schema is used by a publisher to format data for publication and by a reader to validate acquired data and add it to a data collection. Validation ensures that the data conforms to the formatting rules for XML (is well-formed) and to the types, elements, and attributes defined in the schema (is valid). Several schema languages have been proposed for XML; among them XML Schema is the most widely used.

One example of a data provider is the National Center for Biotechnology Information (NCBI).[1] Users can search the NCBI databases to locate data on genes and proteins. The data can then be downloaded in several formats, including as XML. In fact NCBI publishes data in three XML schemas. From an archival or temporal perspective, NCBI like most XML publishers only provides the current *snapshot* of the data. A snapshot is the data that is available at a single point in time, stripped of its historical context. But a data collection varies over time as new data is inserted and existing data is revised. NCBI users can download the current snapshot, but they are unable to track and download changes to data.

In general, scientists want to know the *provenance* of their data: who, what, when, and where [8]; these concerns cut across all scientific data. The evolution of the data is an important part of this provenance. Scientific insights gained by analyzing data often have to be revised when the data changes. To help determine whether a reanalysis is needed, especially in a large data set where manual comparison is infeasible, it is crucial to be able to ascertain whether data has been added, modified, or deleted. One might want to look at coarse changes to an entire XML document or track the evolution over time of specific elements. Depending on the application, it may also be desirable to know the agents responsible for the changes to the data, and if these changes were checked for accuracy.

Let's consider the general process by which a user downloads data from a publisher like NCBI and how a cross-cutting concern (in this case, a temporal aspect) changes that process. Fig. 1 illustrates existing practice. A user requests the current snapshot, D_{now}. The data is then added to the reader's data collection, DB, typically by overwriting a previously acquired version of D in DB. A better strategy is to transmit only the changes to the data, as shown in Fig. 2. A user requests a *change summary* of updates to D from time t, when the user last acquired D, to now. The summary, which is represented as "$\Delta\mathsf{D}$," is used to update the local snapshot of D. The strategy is "better" because the change summary is usually smaller than the current snapshot. NCBI does not currently provide change summaries, but there are platforms that support change summaries such as IBM's Service Data Object (SDO) technology [40]. Unfortunately, neither of these strategies works for scientists that need to track the data's provenance because neither tracks the history of the data. In contrast, Fig. 3 shows the process of acquiring *temporal data*. Temporal data is data with embedded timestamps. The timestamps record the lifetimes of the data items. A user requests a *slice* of data from time t, when the user last acquired D, to now. The slice as returned by the server is represented as "$\Delta\mathsf{D}_{[t,now]}$." The temporal data is then added to DB, extending the history of D.

[1] NCBI: `http://www.ncbi.nlm.nih.gov/`

Unlike the snapshot data in Fig. 1 and Fig. 2, temporal data has embedded metadata that records the entire version history of every data item.

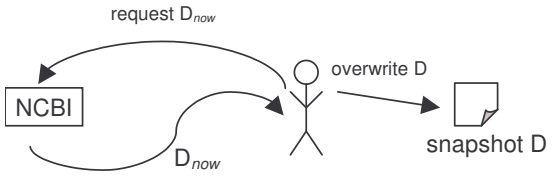


Fig. 1 Download the current snapshot

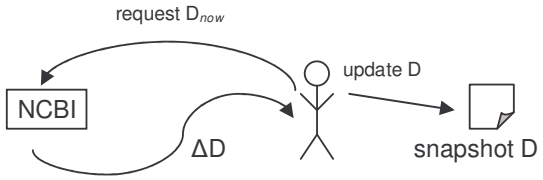


Fig. 2 Download a change summary, e.g., in an SDO

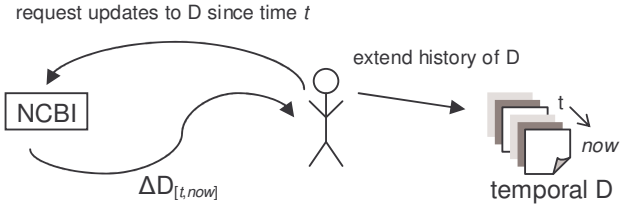


Fig. 3 Download temporal data

Systems that support the publication of and subscription to data with embedded metadata such as timestamps need several novel features.

- A data publisher has to add metadata markup to indicate the *metadata perspective* of versions of the data. The perspective is the metadata that describes an individual version; for temporal metadata it would consist of a timestamp that indicates the lifetime of a version.
- To accommodate the embedded metadata, the schema of the data has to be augmented. Otherwise it would not be possible for the data produced by a publisher to be amenable to automatic processing on the reader's side; for instance, the reader has to be able to validate the data with embedded metadata and update a data store.

- To conserve bandwidth the slice "$\Delta D_{[t\text{-}now]}$" should be compact. Ideally it will be proportional in size to the changes to D since time t.
- It should be possible to validate the changes to a data collection, i.e., the slice "$\Delta D_{[t\text{-}now]}$", separately from the rest of the data collection. Unfortunately an SDO's change summary cannot be validated using the data's schema, rather the changes must first be applied to the data, and the data entirely re-validated. It would be more desirable and less costly if it were possible to validate a slice of data with embedded metadata in isolation from the rest of a data collection.

All of the above features can be supported by weaving a schema aspect into a data schema to produce a schema tapestry that mediates the exchange of XML data. There are some tangential issues that we do not cover in this paper. For instance, a publisher may have changed its schema since time t, so each step in the process must account for changes to the schema as well [38][39].

This paper presents a system to help schema designers develop schemas for data annotated with metadata such as metadata that describes who has access to the data, how the data was measured, and when the data is current, among other aspects. More specifically, we present aspect-oriented XMLSchema (*AOXSchema*) which is an infrastructure and suite of tools for constructing and validating XML data with embedded metadata. AOXSchema extends τXSchema [14][17], adding support for more kinds of metadata than just time. AOXSchema adds *aspect-enhanced element types* to XML Schema. An aspect-enhanced element type denotes that an element can have metadata, describes how to construct and represent the metadata, and provides some simple constraints that broadly characterize how the metadata is used. Aspect-enhanced element types are specified in schema aspects, one aspect per kind of metadata.

An important goal in the development of AOXSchema was to maximally reuse existing XML standards and technology. Biologists are reticent to learn a new data model, or even a significant extension of a data model with which they have just gotten comfortable. Similarly, they do not want to have to acquire and learn how to use a new suite of tools that comes with the new data model. In AOXSchema, any element type can be denoted as an aspect-enhanced element type by annotating it with a single, simple annotation in a schema aspect. The tools operate in most cases identically to extant tools and in fact utilize those existing tools, such as conventional validating parsers. In most cases, the scientists don't even need to care whether their XML data has metadata.

This paper is organized as follows. The next section motivates the differences between conventional XML data and aspect-enhanced XML data. We then discuss how snapshots of a data collection are glued to create *items* and *versions*. The extensions to XML Schema to support schema aspects are pre-

sented in Section 4. Section 5 sketches the process of constructing a *representational schema*, which helps in validating an aspect-enhanced data collection with a schema tapestry. The paper concludes with a discussion of related work and a summary.

2 Example

Assume that data on the gene trypsin 4 (TRY4) is in an XML data collection called `gene.xml`. The collection has information about gene function, which is described using the Mouse Genome Institute ontology.[2] On January 9, 2007 (represented as 2007-01-09) the function of TRY4 was unknown as shown by the XML in Fig. 4. In subsequent months, new scientific data about TRY4 became available. On 2007-02-14 it was learned that TRY4 is involved in synthesizing the trypsinogen protein. The value of the "`function`" attribute was updated creating a new version of the data, as shown in Fig. 5. On 2007-03-06, the gene description became more specific, relating TRY4 to β-cell receptors so an additional "`desc`" element was updated as shown in Fig. 6.

Researchers that prepared a paper on TRY4 in 2007-01-05 would like to learn of any updates to the TRY4 data since that time, and in particular how the data has changed. They would also like to track the *reliability* of the data and of subsequent updates. NCBI imports curated and uncurated data; curated data has been checked for accuracy by trusted experts (e.g., Swiss-Prot is perhaps the best known collection of curated proteomic data[3]). Certain changes will require a new analysis of their experiments. But the data in each figure is the data at a single point in time; the reliability is implicit. Instead of the current snapshot, the researchers need the (*transaction time*) *version history*, which consists of the information in each version of the data along with a timestamp indicating the version's lifetime.[4] Transaction time is the system time when the data was edited. The version history would describe how the knowledge about a particular gene has changed over time. This is of particular interest since new genomic and proteomic data is being constantly generated, and existing data is being revised and corrected. A version history would also aid in time-related analysis such as in tracking how a disease and its symptoms evolve over time (e.g., in an epidemic like the avian flu). The researchers also need to record the data's reliability since they prefer to work only with

[2] Mouse Genome Institute: `http://www.informatics.jax.org`

[3] Swiss-Prot: `http://www.expasy.org/sprot/`

[4] Temporal data could also record the *valid time* versions (valid time is real world time) but for simplicity we consider only one kind of time in this paper, i.e., the transaction and valid times are the same (other relationships between valid and transaction time [25] can be easily modeled in our framework).

curated data (another set of researchers may have a preference—not a requirement—for curated data, but may be content with knowing its origins and level of reliability, which would allow them to make decisions regarding usage).

```
<gene name="TRY4">
    <desc>trypsin 4</desc>
    <ontology ref="MGI" function="unknown"/>
</gene>
```

Fig. 4 `gene.xml` on 2007-01-09, curated data

```
<gene name="TRY4">
    <desc>trypsin 4</desc>
    <ontology ref="MGI"
              function="synthesizes trypsinogen"/>
</gene>
```

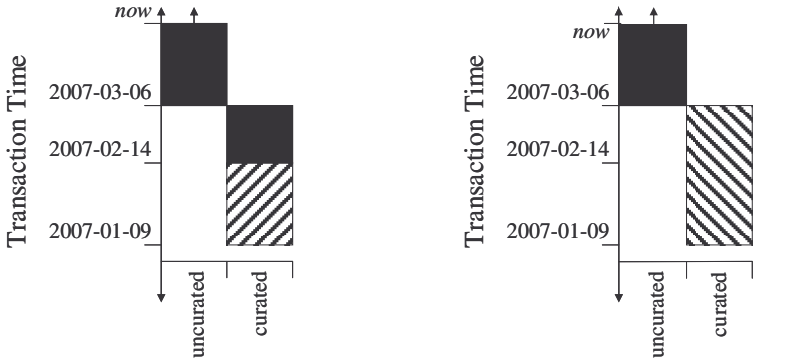
Fig. 5 TRY4 codes for a protein, as of 2007-02-14, curated

```
<gene name="TRY4">
    <desc>trypsin 4, beta-cell receptor</desc>
    <ontology ref="MGI"
              function="synthesizes trypsinogen"/>
</gene>
```

Fig. 6 TRY4 is related to β-cell receptors, as of 2007-03-06, uncurated

To illustrate these changes from the perspective of the metadata, we adopt a technique pioneered by the Bitemporal Conceptual Data Model (BCDM) [26]. The idea is that the metadata creates a multidimensional space, one dimension for each kind of metadata. In this example, there are two dimensions: time and reliability. Within this space it is possible to identify regions where the data remains *constant* or unchanged. Fig. 7a) shows two such regions for the `<ontology>` element. The first region is curated data that begins its lifetime at 2007-01-09 and ends just before 2007-02-14 when the ontology element was changed. In the figure, this region is diagonally striped. The next region is curated data from 2007-02-14 to just before 2007-03-06, and is shaded black in the figure. Interestingly the region switches to a reliability of uncurated from 2007-03-06 until *now*. Note that the shape of a region in the multidimensional space need not be an *orthotope* (an orthotope is an *n*-dimensional generalization of a rectangle, but our regions can be any shape) [13]. Fig. 7b) shows the regions for the `<gene>` element. It also has two regions one diagonally striped and one shaded black in the figure. The first region extends in the transaction time dimension from 2007-01-09 until 2007-03-06 since

<gene> did not change during this time (excluding changes to its <ontology> subelement). On 2007-03-06 the <desc> subelement was modified and the reliability changed. (We will explain in detail later in the paper why <desc> is considered part of <gene> but <ontology> is not when considering changes to <gene>. Basically the schema aspect designer specifies which elements to monitor; in this particular case, the schema aspect designer decided not to track changes to <desc> separately, but rather to couple changes to <desc> with changes to <gene>, while treating changes to <ontology> separately. This has implications, to be explored later in this paper, for how the changes are represented, as well as for schema validation: what changes are allowed.)



a) Metadata regions for the two versions of <ontology>

b) Metadata regions for the two versions of <gene>

Fig. 7 The metadata regions of each data version

Fig. 8 shows the *aspect-enhanced* data that captures the history and reliability of the TRY4 data. The data is largely a list of gene and ontology *items*. An item is an element that *persists* across individual snapshots, where a snapshot is a slice of the data from an *instantaneous* metadata perspective. For instance, Fig. 5 shows the snapshot of curated data as of 2007-02-20. Each item has an itemId attribute that uniquely numbers the item. There is one gene item in the data, and one ontology item. Each item is referenced by an *item reference element* that places it in the context in which it appears in a snapshot of the data. For example, in Fig. 8 the element <ontology$_{Ref}$> references the ontology item, which indicates that some *version* of that item appears within the context of a <gene> element.

```
<data_Root>
  <data><gene_Ref itemRef="1"/></data>
  <gene_Item itemId="1">
    <gene_Version>
      <perspective>
        <time start="2007-01-09" end="2007-03-05"/>
        <reliability curated="yes"/>
      </perspective>
      <gene name="TRY4">
        <desc>trypsin 4</desc>
        <ontology_Ref itemRef="2"/>
      </gene>
    </gene_Version>
    <gene_Version>
      <perspective>
        <time start="2007-03-06" end="now"/>
        <reliability curated="no"/>
      </perspective >
      <gene name="TRY4">
        <desc>trypsin 4, beta-cell receptor</desc>
        <ontology_Ref itemRef="2"/>
      </gene>
    </gene_Version>
  </gene_Item>
  <ontology_Item itemId="2">
    <ontology_Version>
      <perspective>
        <time start="2007-01-09" end="2007-02-13"/>
        <reliability curated="yes"/>
      </perspective>
      <ontology ref="MGI" function="unknown"/>
    </ontology_Version>
    <ontology_Version>
      <perspective>
        <time start="2007-02-14" end="2007-03-05"/>
        <reliability curated="yes"/>
      </perspective>
      <perspective>
        <time start="2007-03-06" end="now"/>
        <reliability curated="no"/>
      </perspective>
      <ontology ref="MGI"
                function="synthesizes trypsinogen"/>
    </ontology_Version>
  </ontology_Item>
</data_Root>
```

Fig. 8 Aspect-enhanced XML data

Whenever the item changes, a new version of the item is created. A change is defined, roughly, as a difference in an element's non-aspect content, exclusive of changes to content within the items that appear as subelements. Hence, the gene item has two versions. The second version was created on 2007-03-06 when new text content was added to the nontemporal `<desc>` element. The `<time>` element in each version's perspective indicates the version's lifetime, while the `<reliability>` element is its reliability. The end time of the second version is "*now*" indicating that the version is current. The ontology item also has two versions, because an attribute value was changed on 2007-02-14. Note that the second version of the ontology item has multiple metadata perspectives.

In general an aspect-enhanced data collection encompasses data from many potential metadata perspectives. For instance, when a temporal aspect is woven into data not only is the current state of the data captured, but all previous versions as well. The aspect embeds timestamps in the data to indicate when each version was current. Hence, a temporal aspect woven into a data collection is unlike an SDO or related technology that records only a single snapshot and/or a summary of changes from a previous version.

One contribution of this paper is a description of how to construct the aspect-enhanced data (Fig. 8) by gluing the data in individual snapshots (Fig. 4, Fig. 5, and Fig. 6), and adding metadata. The data collection in Fig. 8 captures the lifetime of each version [23] as well as its reliability.

Another contribution of this paper is explaining how to compactly represent in XML the change across a number of versions. Though the aspect-enhanced data shown in Fig. 8 appears verbose in this small example, in general, it is actually *compact* in the sense that each edit results in only a localized change to the data (basically, a new version is created within an item). Fig. 9 shows the difference between the first and second versions of the data. The difference is a new version of the ontology element. The ability to represent the difference between two versions in isolation from the rest of the data is useful in both data streaming and refreshing data from a remote source, since the change is usually much smaller in size than the entire collection or even a snapshot. Note that the value of the `itemId` attribute in Fig. 9 is local to the temporal data being exchanged (the value of the attribute could be "`23`") and unrelated to the `itemId` of "`1`" in Fig. 8. A user requesting a change summary since their last access (presumably made between 2007-01-09 and 2007-02-13) can be compactly provided with the subsequent versions (in this case a single version, as seen in Fig. 9).

```
<data_Root>
  <ontology_Item itemId="1">
    <ontology_Version>
      <perspective>
        <time start="2007-02-14" end="2007-03-05"/>
        <reliability curated="yes"/>
      </perspective>
      <perspective>
        <time start="2007-03-06" end="now"/>
        <reliability curated="no"/>
      </perspective>
      <ontology ref="MGI"
                function="synthesizes trypsinogen"/>
    </ontology_Version>
  </ontology_Item>
</data_Root>
```

Fig. 9 The difference between two versions

```
<element name="gene">
  <complexType>
    <attribute name="name" type="text" use="required"/>
    <sequence>
      <element name="desc" type="string"/>
      <element ref="ontology" minOccurs="0"
                              maxOccurs="unbounded"/>
    </sequence>
  </complexType>
</element>
<element name="ontology">
  <complexType>
    <attribute name="ref" type="text"/>
    <attribute name="function" type="text"/>
  </complexType>
</element>
```

Fig. 10 An extract from the gene data schema

```
<element name="gene">
```

```
<aop:item>
  <aop:itemIdentifier>
    <aop:field path="@name"/>
  </aop:itemIdentifier>
  <txs:transactionTime
      kind="state"
      contentVarying="true"
      existenceVarying="no gaps"/>
  <rel:curated/>
</aop:item>
```

definition of gene from the snapshot schema omitted for space

```
</element>
<element name="ontology">
```

```
<aop:item>
  <aop:ItemIdentifier>
    <aop:field path="../@name"/>
    <aop:field path="@function"/>
  </aop:ItemIdentifier>
  <txs:transactionTime
      kind="state"
      contentVarying="true"
      existenceVarying="gaps allowed"/>
  <rel:curated/>
</aop:item>
```

definition of ontology from the snapshot schema omitted for space

```
</element>
```

Fig. 11 An extract from a schema tapestry

A third contribution is a description of a process to construct a schema to validate and interpret the aspect-enhanced data. Typically, the structure of published data is described in an associated schema document. Assume that the file `gene.xsd` contains the *snapshot schema* for `gene.xml`. The snapshot schema is the schema for the data only with no embedded metadata. The snapshot schema is a valuable guide for editing and querying individual snapshots. The snapshot schema is given (in part) in Fig. 10. Note that the schema describes the structure of the fragment shown in Fig. 4, in Fig. 5, and in Fig. 6. Though each snapshot conforms to the schema, the aspect-enhanced data does not. So a snapshot schema such as `gene.xsd` cannot be used (directly)

to validate or interpret the data in Fig. 8. Nor can the schema be used to validate version differences, such as the fragment shown in Fig. 9. In our approach a snapshot schema is woven with schema aspects to create a schema tapestry. In this example there is a temporal aspect and a reliability aspect. These aspects describe, at a logical level, which elements can vary over time and reliability, and how those elements can change. Fig. 11 shows the temporal and reliability aspects as they are woven into the schema tapestry for the running example. There are aspect-oriented annotations for both the gene and ontology element type definitions. The annotations are shown within the grey-lined rectangles in the figure. (Section 4 describes the annotations in detail.) We present the schema tapestry here to emphasize that AOXSchema is fully-upwards compatible with XML Schema; that is, it extends but does not change XML Schema. A further advantage of our approach is that the schema tapestry can also be used to validate the differences between versions, such as the data in Fig. 9.

3 Aspect-Enhanced Data

This section briefly reviews concepts related to aspect-enhanced data and then discusses how to associate elements in different snapshots to create such data.

3.1 A Simple Model for Aspect-Enhanced Data

Let D be an XML data collection. D is typically modeled as an ordered forest, $D = (E, V)$, where E is the set of edges and V is the set of nodes. Each edge in E is of the form (v, w, n) where v is the parent, w, is the child, and n is an ordinal representing the position of the child in the lexical ordering of the children. We will refer to XML data acquired from a document with no embedded metadata as a *snapshot*.

A snapshot is data from a single metadata *perspective*. A perspective is a list of values, $v_1, \ldots, v_n$, where value v_k is chosen from the k^{th} metadata dimension. Each dimension is a set of values. We will assume an ordering of the values when it is conventional to do so. For instance, the transaction time dimension is a set of time points ordered from time `0` (or *`beginning`*) to the current time (*`now`*). The reliability dimension is the set {`uncurated`, `curated`}, with no ordering among the values. So an example metadata perspective is "`3`, `curated`" where "`3`" is chosen from the transaction time dimension and "`curated`" is chosen from the reliability dimension.

Aspect-enhanced data is data with embedded metadata. The metadata describes or modifies the data. Conceptually, the metadata can be embedded in a data forest as follows. Let $D^A = (E^A, V^A)$ be an aspect-enhanced data collection

where E^A is a set of edges and V^A is a set of nodes. Each edge in E^A is of the same form as an edge in a non-aspect-enhanced forest, while each node in V^A is of the form (d, M) where d is the data (as in a non-aspect-enhanced forest) and M is a (possibly empty) aspect-enhanced forest which represents the metadata for d. Consider the modeling of the aspect-enhanced data in Fig. 9. In the forest there is a node corresponding to the ontology element. The node has a data portion, the ontology element itself, and a metadata forest. The forest contains two trees, one for each perspective. As an aside, note that this model supports increasing levels of metadata, i.e., meta-metadata, since a metadata forest could have metadata. But in this paper we consider only a single level.

An aspect-enhanced data collection represents a set of snapshots. Let D^A be an aspect-enhanced data collection. The *snapshot operation* extracts a snapshot of D^A from a specific metadata perspective. Metadata is *not* represented in the snapshot. The snapshot operation is denoted as $snap(p, D^A) = D$ where D is the snapshot from perspective p of D^A. Note that we haven't yet described the structure of aspect-enhanced data; however, it should faithfully capture entire snapshots as stated in the following definition.

Definition [Snapshot reducibility] An aspect-enhanced data collection, D^A, is said to be *snapshot reducible* to snapshots $D_1, \ldots, D_m$ iff $\forall i\ (1 \le i \le m)\ \exists p \in \{ (v_1, \ldots, v_n) \mid$ value v_k is chosen from the k^{th} metadata dimension$\}$ such that $D_i = snap(p, D^A)$.

3.2 Compact Aspect-Enhanced Data

To create compact aspect-enhanced data it is important to identify which elements persist through changes to a data collection. We will sometimes refer to the process of associating elements that persist across various snapshots as *gluing* the elements. When a pair of elements is glued, an *item* is created. An item is an element that evolves through various *versions*. Only aspect-enhanced elements (that is, elements of a type that has an aspect annotation as described further in Section 4) are candidates for gluing.

3.2.1 Item Identifiers

Determining which elements should be glued depends on two factors: the *type* of the element, and the *item identifier* for the element's type. The type of an element is the element's definition in the schema. We will denote the type of an element as $\mathcal{T}$. An element can be glued only to an element or item of the same type. An item identifier is used to identify an item in a snapshot. The identifier is a list of XPath expressions (much like a key in XML Schema) so we first define what it means to evaluate an XPath expression.

Definition [XPath evaluation] Let $eval(x, E)$ denote the result of evaluating an XPath expression E from a context node x. Given a list of XPath expressions, $L = [E_1, \ldots, E_k]$, then $eval(x, L) = [eval(x, E_1), \ldots, eval(x, E_k)]$.

Since an XPath expression evaluates to a list of values, $eval(x, L)$ evaluates to a list of lists. An item identifier is a list of XPath expressions.

Definition [Item identifier] An *item identifier* for an aspect-enhanced type, $\mathcal{T}$, is a list of XPath expressions, $L_{\mathcal{T}}$, such that for each element x of type $\mathcal{T}$, $eval(x, L_{\mathcal{T}})$ names the *item* to which x belongs.

Each item identifier is specified by a schema designer. (Elsewhere we sketch a method for automatically constructing item identifiers utilizing historical information from a varying instance [44]. In the present paper, we encourage the schema designer to specify the item identifiers, as that is a component of the semantics of the underlying data.) Usually each item identifier will be the (snapshot) key for the element type given in the schema [9]. But an item identifier may differ from a snapshot key since the identifier should be a temporally-invariant key [33].

Example [Item identifiers] As an example, a designer might specify the following item identifiers for the aspect-enhanced elements in Fig. 8.

- `<gene>` → [`@name`]
- `<ontology>` → [`../@name`, `@function`]

The item identifier for a `<gene>` is the name of the gene while the item identifier for an `<ontology>` is the gene's name (its parent's item identifier) combined with the gene's `function` attribute value.

Items represent semantic clusters: information that is lumped together and logically changes as a unit. We saw in Fig. 9 an example of a difference between two versions; this difference is expressed in terms of the item(s) that changed. So a schema designer will designate how they wish to gather the information about changes, or equivalently, how they think the changes will be clustered. If an `<ontology>` was considered part of the <gene>, then any change to the `<ontology>` would be considered a change to the `<gene>`. In the example above a schema designer is saying that conceptually an `<ontology>` is a component or feature of a `<gene>` that changes (in terms of content, reliability, or whatever aspect) separately from the `<gene>` itself (which can also change, e.g., its description). Clearly this determination is subjective. The schema designer will want to define a small number of items, and will probably not want to make every element type a separate item. Also, it should be emphasized that an item's aspect(s) are orthogonal: an item can be associated independently with each aspect.

We will further restrict item identifiers to be *unique* within a snapshot, that is, at most one element in each snapshot can belong to an item.

3.2.2 Building Items

Once an item has been identified, the next step is to determine whether an item's content changes or remains the same across different snapshots. Potentially if an item's content remains the same over two snapshots then the content can be combined to create a compact representation (more compact than representing the same content again and again). Snapshot elements that are "adjacent" and "the same" can be compacted and associated with a region of metadata, as we did in Fig. 7. These compacted elements form versions. So an item is a set of versions, as defined below.

Definition [Item] Let **item**(x) be the item named by $eval(x, L_{\mathcal{T}})$ where x is of type $\mathcal{T}$. Then **item**(x) = $\{(v_1, p_1), \ldots, (v_n, p_n)\}$ where each v_i is a *version* of x with *perspective* p_i ($1 \le i \le n$).

3.2.3 Creating Versions

A version is a copy of the subtree rooted at the item, where each branch in the copy terminates at a leaf (attribute node, text node, etc.) or at the first element on the branch that is associated with some other item, which is replaced with an *item reference*.

Definition [Version] Let **item**(x) be an item of type $\mathcal{T}$ in snapshot $D = (E, V)$. Let (E_x, V_x) be the subtree rooted at x in D. Then version v of x is (E_v, V_v) where

$E_v = \{(a_v, b_v, n) \mid (a_x, b_x, n) \in E_x \wedge (b_x \text{ is an item} \Rightarrow b_v \text{ is an } \textit{item reference})$
$\wedge (a_x \text{ is an item} \Rightarrow a_v = x) \wedge (a_x \text{ and } b_x \text{ are not items} \Rightarrow a_v = a_x \wedge b_v = b_x)\}$

and $V_v = \{v \mid (v, \_, \_) \in E_x \vee (\_, v, \_) \in E_x\} \cup \{x\}$.

Example [Items] Items appear throughout the example of aspect-enhanced data shown in Fig. 8. The first version of the `<gene>` item is a copy of the `<gene>` element in Fig. 4, which is the first snapshot of the data. Note that the `<ontology>` element is an item, so it has been replaced in Fig. 8 by an item reference whereas the `<desc>` element is unchanged since it is not an item.

Versions that are adjacent and the same can be compacted in an item to reduce the size of the representation. The kind of metadata plays an important role in determining adjacency. Below we define adjacency for an ordered dimension (e.g., transaction time) and then an unordered dimension (e.g., reliability). Partially ordered dimensions can also be handled.

Definition [Adjacent in an ordered metadata dimension] Without loss of generality, let there be a single ordered metadata dimension (so the perspective is a single value). Let x be an element of type $\mathcal{T}$ in $snap(i, D^A)$. Let y be an element of type $\mathcal{T}$ in $snap(j, D^A)$. Finally let $L_{\mathcal{T}}$ be the item identifier for elements of type $\mathcal{T}$. Then x is *metadata adjacent* to y if and only if $eval(x, L_{\mathcal{T}}) = eval(y, L_{\mathcal{T}})$ and it is not the case that there exists an element z of type $\mathcal{T}$ in a snapshot between (exclusive) the i^{th} and j^{th} snapshots such that $eval(z, L_{\mathcal{T}}) = eval(x, L_{\mathcal{T}})$.

Adjacency in an unordered dimension is very straightforward, basically everything is adjacent.

Definition [Adjacent in an unordered metadata dimension] Without loss of generality, let there be a single ordered metadata dimension (so the perspective is a single value). Let x be an element of type $\mathcal{T}$ in $snap(i, D^A)$. Let y be an element of type $\mathcal{T}$ in $snap(j, D^A)$. Finally let $L_{\mathcal{T}}$ be the item identifier for elements of type $\mathcal{T}$. Then x is *metadata adjacent* to y if and only if $eval(x, L_{\mathcal{T}}) = eval(y, L_{\mathcal{T}})$.

When multiple metadata dimensions are present, two elements are considered adjacent only if they are adjacent in every dimension.

Recall that a version is a compacted element, where adjacent elements that are the same are represented only once. "Sameness" is observed within the context of the Document Object Model (DOM).

Definition [DOM equivalence] A pair of item versions is *DOM equivalent* if the pair meets all of the following conditions: they have the same number of children, same element tag, same set of attributes (an attribute is a name, value pair), and same text content, and for each child, the child is DOM equivalent to the corresponding child of the other (in a lexical ordering of the children).

As an aside, we observe that DOM equivalence in an AOP XML context is akin to *value equivalence* in a temporal relational database context [23].

A version is associated with a metadata perspective coalesced from the various snapshots. The perspective of a version captures the set of metadata conditions for which the version's content is valid. A version's perspective is extended when adjacent versions are DOM equivalent (the perspective can have gaps or holes, although having a gap may violate a schema constraint as described in Section 4). A new version is created when adjacent versions in the same item are *not* DOM equivalent.

Definition [Version creation/extension] Let $\mathbf{item}(x) = \{(v_1, p_1), \ldots, (v_n, p_n)\}$. Let y be a new version, (w, q), of $\mathbf{item}(x)$, that is y has the same type and item identification as $\mathbf{item}(x)$. If there exists version $(v_i, p_i) \in \mathbf{item}(x)$ that is DOM equivalent and adjacent in *every* metadata dimension to y then replace (v_i, p_i) with $(v_i, \mathbf{coalesce}(q, p_i))$. Otherwise add (w, q) to $\mathbf{item}(x)$.

The **coalesce** operation merges two metadata perspectives. The specific operation to use for coalescing metadata depends on the kind of metadata. For an unordered metadata domain, **coalesce** is set union, for an ordered domain it is interval union (e.g., timestamps are coalesced to create temporal elements [20]).

Example [Versions] Fig. 12 depicts the items and versions in the example in Section 2. An abstract representation of the DOM for each snapshot of the data is shown. The items in the sequence of snapshots are connected within each grey shaded region. There is one gene item and one ontology item. Each item has two versions. The transition between versions is shown as a black rectangle on the grey

connection arcs. The gene item has a new version when the content of the <desc> element changes and the ontology item has a new version when its content is modified on 2007-02-14.

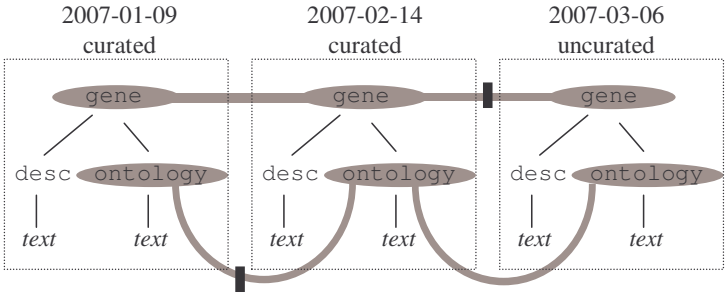


Fig. 12 Items and versions in the example

4 XML Schema Extensions

In this section we present the few extensions to XML Schema to support aspect-oriented schema design. The presentation has three parts. First we develop an architecture for supporting AOP concepts. Next we sketch the design of an aspect, focusing on a temporal aspect. Finally, we show how to specify schema cut points to weave the schema aspects into a tapestry. The overarching design goal in all of these steps is to use XML Schema rather than replace or modify it so we chose to keep AOXSchema consistent with the XML Schema standard.

4.1 Architecture

The architecture of AOXSchema is illustrated in Fig. 13. 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 grey in the figure, that is the snapshot schema, schema aspects, and snapshot data, need to be supplied by a designer. Often a designer can reuse part of a schema aspect (only the cut points need to be specified anew as described in detail in Section 4.2.2).

There is one schema aspect for each kind of metadata. In our running example there would be a schema aspect for transaction time and one for reliability. The schema aspects together with the snapshot schema are fed into the schema weaver to create a schema tapestry. A second tool, the snapshot gluer, takes a collection of snapshots and a schema tapestry to produce an aspect-

enhanced data document. Each of the snapshots imports the snapshot schema, and we implicitly assume that there is metadata associated with each snapshot; this metadata is not shown in the figure. In the running example, the metadata would consist of a transaction time and a reliability for each piece of snapshot data. The aspect-enhanced document can be validated using a special validating parser. The parser is a conventional validating parser augmented with a second phase that validates the constraints uniquely specified for each aspect.

That's an overview of the architecture. We explore each of the pieces in more detail in the remainder of this section.

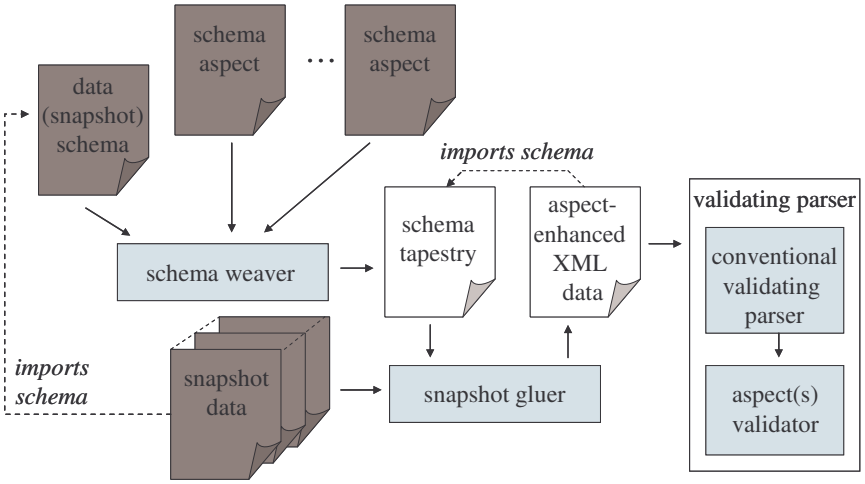


Fig. 13 Design and tool architecture

4.2 Designing a Schema Aspect

AOXSchema extends XML Schema with annotations to denote aspect-enhanced element types, but otherwise leaves XML Schema unchanged. The annotations are made in the relevant schema aspect (such as the temporal schema aspect or the reliability schema aspect seen in Fig. 13). Each of the annotations is described in more detail below.

4.2.1 Items

The annotation is an `<aop:item>` element. The annotation denotes that elements of that type can be items. The `aop` namespace indicates that the annotation is part of AOXSchema. As mentioned previously, items are XML elements that persist across snapshots. An identifier needs to be defined for each item to enable gluing or connecting the elements from different snapshots

(particularly given that much new data might be introduced between snapshots, and the same <gene> element could look quite different across snapshots). Therefore, within an <aop:item> element there must appear an item identifier. Such an identifier has the following general form.

```
<aop:item>
  <aop:itemIdentifier>
     <aop:field path="XPath expression"/>
     …
     <aop:field path="XPath expression"/>
  </aop:itemIdentifier>
  …
</aop:item>
```

An item identifier is list of fields, each of which is a (relative) XPath path expression. Once an item is defined, it is further annotated with aspect specific constraints.

4.2.2 Cut points and SchemaPath

An item annotation can appear in a schema aspect. It is linked to an element type definition in a snapshot schema with an <aop:cutPoint> element. A cut point has the following general form.

```
<aop:cutPoint target="SchemaPath expression">
   <aop:item …/>
    …
</aop:cutPoint>
```

Each cut point has a "target" attribute that designates the location of an element type definition in a snapshot schema. The effect of introducing a cut point is that it determines which aspect (temporal, reliability, etc.) is relevant to an item. More than one aspect could be relevant to an item. In that case, the cut point should be the same in each relevant aspect since there must be agreement on how an item is identified.

The value of the target attribute is a *SchemaPath expression*. SchemaPath is a language for locating an element type definition in a snapshot schema; it is a reduced form of XPath (essentially XPath--). SchemaPath supports only four axes: parent, child, self and attribute, does not have any predicates, and has only a restricted set of node tests. SchemaPath's data model is a graph (rather than a tree) that is created by parsing an instance of a schema. The data model is created as follows. Each element and attribute definition is a node, and a "child" edge is added from a node to each node that represents a possible sub-element of the node. A recursive sub-element (which is possible in XML Schema) will introduce a cycle in the graph. There is also a special "attribute" edge from a node to each attribute for that node. As an example, Fig. 14 shows the graph for the schema in Fig. 10. The dashed lines in the figure

are attribute edges. Though the evaluation of an XPath expression on a graph with cycles might not terminate, a SchemaPath expression will always terminate since SchemaPath axes contain nodes that are at most one edge away from a context node and any SchemaPath expression has only a finite number of axes.

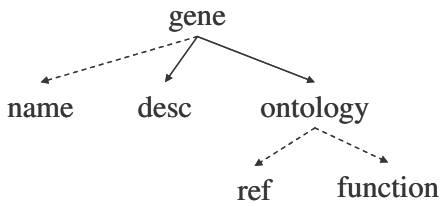


Fig. 14 A SchemaPath instance for the example snapshot schema

Example [Cut points] The biologists would like to capture the transaction time history of `<gene>` and `<ontology>` elements. So they create a temporal aspect. Within the temporal aspect they specify the following cut points.

```
<aop:cutPoint target="/gene">
   <aop:item …/>
    …
</aop:cutPoint>
<aop:cutPoint target="/gene/ontology">
   <aop:item …/>
    …
</aop:cutPoint>
```

While schema aspects are developed in isolation, they are woven together to create a schema tapestry. If an element type definition is annotated by multiple aspects, the item identifier in each aspect for that definition must be the same, or the schema weaver will report an error.

4.2.3 Aspect-Specific Constraints

Additional constraints in a schema are optional. The constraints are evaluated after an item is glued. The constraints are separately specified for each aspect. In this paper we focus only on transaction time. Let's consider a temporal aspect. The constraint specification for a temporal element has the following general form, though constraints on other aspects including reliability may also be specified (e.g., restrictions on the source or curator agents and their roles as described in provenance literature [37]).

```
<txs:transactionTime
   txs:kind="state (default) | event"
   txs:contentVarying="false (default) | true"
   txs:existenceVarying="false | gaps allowed (default) | no gaps"/>
```

The `kind` attribute specifies whether the lifetime of an item has duration; a *state* kind of annotation implies continuity, while an *event* signifies that the lifetime is a single instant. The terminology is borrowed from temporal databases where events occur at a single instant in time (e.g., a wedding on July 14, 2007), whereas a state occurs over a period of time (e.g., married from July 14, 2007 until now) [23]. The `contentVarying` attribute is used to specify whether an item's content must be constant over time, or can vary. The `existenceVarying` attribute governs whether the element corresponding to a particular item can come and go in various snapshots. If the value of the attribute is *false*, then the underlying element must be in every snapshot (or never appear). If the existence is *no gaps*, then once the element has been deleted from a snapshot, it cannot reappear in a later snapshot. Otherwise, an item's existence is unrestricted. Each attribute is optional, as is the transaction time element. If the attribute is not specified, the indicated default value applies.

Example [AOXSchema] The biologists in our running example are interested primarily in tracking two kinds of changes to the NCBI data: revisions of the gene itself and revisions of the ontology elements. Since NCBI does not publish a temporal schema, biologists must download individual snapshots and maintain a temporal data collection locally. Towards this end they create the annotations given in Fig. 11. The gene and ontology element type definitions given in the snapshot NCBI schema are annotated to indicate that they are items, and so a version history will be kept for each element of those types. While genes can be both content and existence varying, a gene's existence is slightly constrained to disallow gaps since once a gene is discovered, it is not deleted and later "re-discovered". Therefore this constraint specifies that in order for the data to be valid a gene cannot be deleted and then (later) reinserted.

Currently, the temporal aspect in AOXSchema has a restricted set of temporal constraints. Richer classes of temporal constraints have been proposed [3][12][15], but for simplicity and brevity we limit the variety of constraints in the current system.

4.2.4 The Snapshot Gluer Tool

The snapshot gluer produces an aspect-enhanced data collection from a set of snapshots using a schema tapestry. The algorithm for the snapshot gluer for one dimension is outlined in Fig. 15. Recall that aspect-enhanced data is a set of items, where each item is a sequence of versions and each version is a data element paired with a metadata perspective. So the first task for the gluer is to identify items. Towards this end, the gluer evaluates the item identifiers in the tapestry for every snapshot to find the elements that belong to each item. Next, the gluer forms versions by partitioning the set of elements for an item into DOM equivalent subsets. Finally the metadata for each partition is coa-

lesced as discussed in Section 3.2 to reduce the size of the representation. The gluing and coalescing can be done in separate passes for each aspect or for every aspect simultaneously; the final result will be the same in either case.

Input – Schema tapestry and a set of snapshots.
Output – An aspect-enhanced data collection.
Data structures used – Hash table. An item identifier of an item is used as a hash key. The item is the hash value.
Algorithm:

```
for every snapshot in the set of snapshots do
  for every element in the snapshot do
    if the element type definition is present in a schema aspect
      evaluate the item identifier
      if the identifier is in the hash table
        if the element is DOM equivalent to some version in the item
          coalesce the metadata with the version
        else create a new version
      else create a new item in the hash table, with one version
```

Fig. 15 The snapshot gluer algorithm

The time cost of the algorithm is modest. Let S be the number of snapshots, E be the number of elements in a snapshot (that is the size of the snapshot), A be the number of aspects, V be the number of versions in an item, and M be the size of the metadata associated with a version. The algorithm iterates through the snapshots and elements within each snapshot; these nested loops cost $O(S*E)$. Each time through the inner loop up to $O(A)$ item identifiers are evaluated, and at most $O(V)$ versions created. For each version the metadata must be coalesced, which costs $O(M^2)$ since the metadata in the element must be associated with the metadata in the version. Hence the total cost of the algorithm is $O(S*E*(A+V+M^2))$. We anticipate that in practice A and M will be small, so the cost devolves to $O(S*E*V)$.

5 The Representational Schema

The representational schema is a conventional XML Schema document that is automatically generated from an AOXSchema document. It is used to validate aspect-enhanced data using a conventional validating parser. This section describes how to convert the schema tapestry to a *representational schema*. A representational schema is used to validate the representation of the aspect-

enhanced data. The representational schema is transitory; it is needed only for validation, and in fact need never be seen by the user.

An XML Schema specification can be viewed as a grammar. The grammar consists of productions of the following form for each element type.

$S \rightarrow$ <s> α </s>

In the above production, α describes the content of elements of type *S*.

A schema aspect specifies that some of the element types are aspect-enhanced. To construct a representational schema, several new productions are added to the snapshot schema for each aspect-enhanced element type; no productions are removed from the snapshot schema though some are modified. Since only elements can be aspect-enhanced, this section focuses on the element-related components of a schema. The construction process consists of several steps. We'll illustrate the process by describing what is done for a single, representative aspect-enhanced element type, *S*.

The first step is to add a production to indicate that the element type *S* is aspect-enhanced, that is, it could be an item. In situ representations of items are replaced by references to that item. The aspect-enhanced production has following form:

$S_{Ref} \rightarrow$ <s*Ref* itemRef="*m*"/>

where <s*Ref*> denotes an aspect-enhanced element of type *S* and `itemRef` is a reference to an item of type *S*.

Next a production is added to define the *S* item type.

$S_{Item} \rightarrow$ <s*Item* itemId="*n*"> $S_{Version}$+ </s*Item*>

An item has a unique `itemId` value, and consists of a list of *versions*.

The third step is to add a production to specify each version of type *S*. The production for a version of an element of type *S* has the following form:

$S_{Version} \rightarrow$ <s*Version*>
<perspective> ρ </perspective>
S
</s*Version*>

where ρ is the schema of each aspect's perspective and *S* is the snapshot definition of the element's type. The perspective in a version records the metadata conditions for which the version is valid. We do not impose a particular schema for an aspect's perspective, rather we assume that the schema is given separately in an aspect and woven into the schema tapestry. Without loss of generality we will assume for the aspects in this paper that each perspective has the following form.

ρ → <time start="…" end="…"/>
<reliability curated="…"/>

The next step is to modify the *context* in which an aspect-enhanced element appears. For each aspect-enhanced element type, S, that appears in the right side of a production, replace S with S_{Ref}. For example, assume that the schema has a production of the following form:

$X \rightarrow$ <x> β S γ </x>

where β and γ describe arbitrary content before and after S, respectively. The production is replaced by the following production.

$X \rightarrow$ <x> β S_{Ref} γ </x>

Only the element type is replaced, any other constraints on the element are kept (e.g., minoccurs and maxoccurs are unaffected). This process is repeated for every aspect-enhanced element type.

The final step is to augment the root element type with an additional production that appends a list of items. Let the root be an element of type R. Then the new root becomes the following.

$R_{Root} \rightarrow$ <data$_{Root}$> R? X_{Item}* </data$_{Root}$>

where X_{Item} is a list of item types. The production for X_{Item} is given below, where each S^i_{Item} is one of k item types.

$X_{Item} \rightarrow S^1_{Item} \mid \ldots \mid S^k_{Item}$

An additional step is needed to recast constraints that appear in the original schema. One such constraint is the uniqueness constraint imposed by a DTD identifier or XML Schema key definition. Since the same identifiers and key values can appear in multiple versions of an element, such values are no longer unique in an aspect-enhanced data collection, even though they are unique within each snapshot. In temporal relational databases, the concept of a *temporal key*, which combines a snapshot key with a timestamp, has been introduced. Temporal keys can be enforced by a temporal validating parser, but not by a conventional parser. So constraints that impose uniqueness within a snapshot must be relaxed or redefined as follows. The value of each `id` type attribute in an aspect-enhanced element is rewritten to be a unique value; `idRef`s are similarly rewritten. Finally, schema keys are rewritten to include `itemId`s and perspectives, creating a key more like a temporal key.

It is important to note that the production for the root of the aspect-enhanced data specifies that it is just a list of items. This enables aspect-enhanced data to be incrementally validated, which is critical in data streaming applications.

Example [Representational schema construction] Let's go through the construction process with an example. Assume that the productions in the schema for the example fragment in Fig. 6 are given below.

$R \rightarrow$ <data> G+ </data>
$G \rightarrow$ <gene> D [N | *text*]* </gene>
$D \rightarrow$ <desc> *text* </desc>
$N \rightarrow$ <ontology ref="*text*"> *text* </ontology>

Next, assume that the <gene> and <ontology> element types are aspect-enhanced, as shown in Fig. 11. The schema would be transformed as follows. First, productions are added for the aspect-enhanced elements.

$G_{Ref} \rightarrow$ <gene$_{Ref}$ itemRef="*n*"/>
$N_{Ref} \rightarrow$ <ontology$_{Ref}$ itemRef="*m*"/>

Next, productions are added for the items.

$G_{Item} \rightarrow$ <gene$_{Item}$ itemId="*n*"> $G_{Version}$+ </gene$_{Item}$>
$N_{Item} \rightarrow$ <ontology$_{Item}$ itemId="*m*"> $N_{Version}$+ </ontology$_{Item}$>

Productions are then added for each version type, and for the perspective in each version.

$G_{Version} \rightarrow$ <gene$_{Version}$>
<perspective> ρ </perspective>
G
</gene$_{Version}$>
$N_{Version} \rightarrow$ <ontology$_{Version}$>
<perspective> ρ </perspective>
N
</ontology$_{Version}$>
ρ → <time start="…" end="…"/>
<reliability curated="…"/>

The next step is to modify the context (i.e, the right side of productions) in which an item could potentially appear.

$R \rightarrow$ <data> G_{Ref}+ </data>
$G \rightarrow$ <gene> D [N_{Ref} | *text*]* </gene>

Finally, the root is modified to include the items.

$R_{Root} \rightarrow$ <data$_{Root}$> R? [G_{Item} | N_{Item}]* </data$_{Root}$>

6 Related Work

Various XML schema specification languages have been proposed in the literature and in the commercial arena. We chose to extend XML Schema be-

cause it is backed by the W3C and supports most major features available in other XML schemas [30]. 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 to other data models, such as UML [35]. As an example, we have extended the Unifying Semantic Model, a conceptual model similar to the ER Model, to utilize annotations [28] very similar to what we propose here.

The introduction pointed to related work in possible aspects, such as annotations, provenance, lineage, and accuracy of data. Research related to a temporal aspect, that is, the representation of temporal data and documents on the web is the most extensive. Grandi has created a bibliography of previous work in this area [22]. Marian et al. [31] discuss versioning to track the history of downloaded documents. Chien, Tsotras and Zaniolo [10] have researched techniques for compactly storing multiple versions of an evolving XML document. Buneman et al. [9] provide another means to store a single copy of an element that occurs in many snapshots. This paper differs from all of the above papers since our focus is on schemas and validation.

It is possible to capture transaction time information for documents through change analysis, as discussed below. Cho and Garcia-Molina [11] provide evidence that some web resources change frequently (though not specifically XML resources). Nguyen et al. [34] describe how to detect changes in XML documents that are accessible via the web [42]. Dyreson et al. [16] describe how a web server can capture some of the versions of a time-varying document. Yu and Popa provide an algorithm to convert either a list of changes or just the original and altered schema to a (more semantic) evolution mapping [43].

Recently there has been interest in incremental validation of XML documents [2][36]. AOXSchema takes an orthogonal approach to incremental validation in so far as the changes to documents can be validated in isolation.

Two papers have previously addressed the issue of validating even temporal data [14][17]. In previous work we developed the τXSchema data model and architecture. In this paper we generalize the architecture to consider all kinds of metadata, not just temporal metadata, and introduce aspects.

AOXSchema focuses on *instance versioning* rather than *schema versioning* [21][38]. The schema describes 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. In other work we consider schema versioning [18].

One final area of related work is *intensional XML data* (also termed dynamic XML documents [1]), that is, parts of XML documents that consist of programs that generate data [32]. Incorporating intensional XML data is beyond the scope of this paper.

7 Conclusion

This paper presents *AOXSchema*, an upward-compatible extension of XMLSchema that refines and extends the support for valid and transaction time in τXSchema with the notion of orthogonal *aspects*. The impetus for this paper arose from an observation by a colleague, Christian S. Jensen, about the nature of temporal data. Christian observed that temporal data is metadata since it modifies a property of the data, such as its validity. To provide more general support for metadata, we adapted techniques from aspect-oriented programming to improve the design of schemas for data with embedded metadata. AOP was developed to quickly and safely add cross-cutting concerns to any program. Data, like programs, also has cross-cutting concerns such as versioning and security. And, like in programs, these concerns are poorly supported in schema design. To add a cross-cutting concern like versioning to a schema, a designer currently has to resort to an ad hoc design. Our goal is to help schema designers easily convert existing snapshot schemas to aspect-enhanced schemas for the construction, management, and validation of data and documents with embedded metadata.

In our approach a cross-cutting schema concern is implemented in a schema aspect. Several such aspects can be woven together with a snapshot schema into a *schema tapestry*. The tapestry has annotations to denote that some element types will have metadata. Each annotation includes an item identifier, which is used to glue elements, yielding an item. Each change in an item over time creates a new version of the item. AOXSchema provides a gluer tool to construct an aspect-enhanced data collection. The system also has a special validating parser. The parser combines a conventional validating parser with aspect-specific constraint checkers. To validate an aspect-enhanced document, a schema tapestry is first converted to a *representational* schema, which is a conventional XML Schema document that describes how the data and embedded metadata is represented. The representational schema is carefully constructed to ensure that every snapshot conforms to the snapshot schema.

The architectural design of the infrastructure and even of the schema language itself is driven by the critical requirement from biologists, and indeed, from data users generally, of *upward compatibility*, of data, of schemas, and even of tools and infrastructure, in the support of data with embedded metadata. This paper has demonstrated how a schema for data can be extended very simply from a snapshot schema, and then how the data manipulation, principally gluing and validation of such data and schema, can be done, utilizing conventional, well-understood tools.

We have designed and implemented tools (including the schema weaver, snapshot gluer, and validating parser illustrated in Figure 13) for transaction

time. These tools show that our approach works and has the advantages listed above. We are now extending these tools to support a second aspect, valid time, which involves considering interaction between multiple aspects, as a schema may just contain transaction time, just contain valid time, or be bitemporal, that is, contain both transaction and valid time. Our next task is to generalize these tools to permit additional aspects to be incorporated in an orthogonal, plug-compatible manner. We plan to then evaluate our approach via the reliability aspect.

Subsequently, we plan to integrate AOXSchema with an XML-based editor. By incorporating AOXSchema, an editor should be able to provide improved revision control and a *change tracking* feature. We have done this for an editor for the afore-mentioned temporal USM conceptual model [29]; it turns out that the upward-compatibility of the language design extends even to design support environment. 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 into pieces such that elements with similar "rates of change" are together [24].

Acknowledgements

The second author acknowledges partial support from NSF grants IIS-0415101, IIS-0639106, and EIA-0080123 and from a grant from Microsoft Corporation.

References

[1] Serge Abiteboul, Angela Bonifati, Grégory Cobéna, Ioana Manolescu, and Tova Milo, "Dynamic XML Documents with Distribution and Replication," in *SIGMOD*, 2003. San Diego, CA. pp. 527–538.

[2] Denilson Barbosa, Alberto O. Mendelzon, Leonid Libkin, Laurent Mignet, and Marcelo Arenas, "Efficient Incremental Validation of XML Documents," in *ICDE*, 2004. Boston, MA, pp. 671–682.

[3] Elisa Bertino, Claudio Bettini, Elena Ferrari and Pierangela Samarati. *An Access Control Model Supporting Periodicity Constraints and Temporal Reasoning*. ACM Transactions on Database Systems, 1998. **23**(3): 231–285.

[4] Deepavali Bhagwat, Laura Chiticariu, Wang Chiew Tan, and Gaurav Vijayvargiya, "An Annotation Management System for Relational Databases," in *VLDB*, 2004. Toronto, CA, pp. 900–911.

[5] Rajendra Bose and James Frew, *Lineage Retrieval for Scientific Data Processing: A Survey*. ACM Computing Surveys, 2005. **37**(1): 1–28.

[6] Peter Buneman, Adriane Chapman, and James Cheney, "Provenance Management in Curated Databases," in *SIGMOD*, 2006. Chicago, IL, pp. 539–550.

[7] Peter Buneman, Sanjeev Khanna, Keishi Tajima, and Wang Chiew Tan, *Archiving Scientific Data*. ACM Transactions on Database Systems, 2004. **29**(1): 2–42.

[8] Peter Buneman, Sanjeev Khanna, and Wang Chiew Tan, "Why and Where: A Characterization of Data Provenance," in *ICDT*, 2001. pp. 316–330.

[9] Peter Buneman, Susan B. Davidson, Wenfei Fan, Carmem S. Hara, and Wang Chiew Tan, *Keys for XML.* Computer Networks, 2002. **39**(5): 473–487.

[10] Shu-Yao Chien. Vassilis J. Tsotras. Carlo Zaniolo, *Efficient Schemes for Managing Multiversion XML Documents*. VLDB Journal, 2002. **11**(4): 332–353.

[11] Junghoo Cho and Hector Garcia-Molina, *Estimating Frequency of Change*. ACM Transactions on Internet Technology, 2003. **3**(3): 256–290.

[12] Jan Chomicki, *Efficient Checking of Temporal Integrity Constraints Using Bounded History Encoding*. ACM Transactions on Database Systems, 1995. **20**(2): 149–186.

[13] H. S. M. Coxeter, *Regular Polytopes*, 3^{rd} *ed*. Dover Publishers, New York, 1973.

[14] Faiz Currim, Sabah Currim, Curtis E. Dyreson, and Richard T. Snodgrass, "A Tale of Two Schemas: Creating a Temporal XML Schema from a Snapshot Schema with τXSchema," in *EDBT*. 2004, pp. 348–365.

[15] Faiz Currim and Sudha Ram, *Conceptually Modeling Windows and Bounds for Space and Time in Database Constraints*. Communications of the ACM, to appear 2007.

[16] Curtis E. Dyreson, Hui-Ling Lin, and Yinxia Wang. "Managing Versions of Web Documents in a Transaction-time Web Server," in *WWW*, 2004. New York, NY. pp. 422–432.

[17] Curtis E. Dyreson, Richard T. Snodgrass, Faiz Currim, and Sabah Currim, "Schema-Mediated Exchange of Temporal XML Data," in *ER*, 2006. Tucson, AZ, 212–227.

[18] Curtis E. Dyreson, Richard T. Snodgrass, Faiz Currim, Sabah Currim, and Shailesh Joshi, "Validating Quicksand: Schema Versioning in τXSchema," in *XSDM* (an *ICDE* Workhsop), 2006. Atlanta, GA, p. 82.

[19] Tzilla Elrad, Robert E. Filman, Atef Bader, *Aspect-Oriented Programming: Introduction*. Communications of the ACM, 2001. **44**(10): 29–32.
[20] Shashi K. Gadia and Jay H. Vaishnav. "A Query Language for a Homogeneous Temporal Database," in *PODS*, 1985, pp. 51–56.
[21] Fabio Grandi, "SVMgr: A Tool for the Management of Schema Versioning," in *ER*, 2004, pp. 860–861.
[22] Fabio Grandi, *An Annotated Bibliography on Temporal and Evolution Aspects in the WorldWideWeb*. 2003, TimeCenter Technical Report.
[23] Christian S. Jensen and Curtis E. Dyreson (eds.), "The Consensus Glossary of Temporal Database Concepts – Feb. 1998 Version," in *Temporal Databases*, 1998. pp. 367–405.
[24] Christian S. Jensen and Richard T. Snodgrass, *Semantics of Time-Varying Information.* Information Systems, 1996. **21**(4): 311–352.
[25] Christian S. Jensen and Richard T. Snodgrass, *Temporal Specialization and Generalization.* IEEE Trans. on Knowledge and Data Engineering, 1994. **6**(6): 954–974.
[26] Christian S. Jensen, Michael D. Soo, Richard T. Snodgrass, "Unification of Temporal Data Models," in *ICDE*, 1993. pp. 262–271.
[27] Anastasios Kementsietsidis, Floris Geerts, and Diego Milano, "MONDRIAN: Annotating and Querying Databases through Colors and Blocks," in *ICDE*, 2006, p. 82.
[28] Vijay Khatri, Sudha Ram, and Richard T. Snodgrass, *Augmenting a Conceptual Model with Geospatiotemporal Annotations.* IEEE Transactions on Knowledge and Data Engineering, 2004. **16**(11): 1324–1338.
[29] Vijay Khatri, Sudha Ram, and Richard T. Snodgrass, *On Augmenting Database Design-Support Environments to Capture the Geo-Spatio-Temporal Data Semantics*. Information Systems, 2005: 1–37.
[30] Dongwon Lee and Wesley W. Chu, *Comparative Analysis of Six XML Schema Languages.* SIGMOD Record, 2000. **29**(3): 76–87.
[31] Amélie Marian, Serge Abiteboul, and Laurent Mignet, "Change-Centric Management of Versions in an XML Warehouse," in *VLDB*, 2001. Rome, Italy, pp. 581–590.
[32] Tova Milo, Serge Abiteboul, Bernd Amann, Omar Benjelloun, and Fred Dang Ngoc, "Exchanging Intensional XML Data," in *SIGMOD*, 2003. San Diego, CA, pp. 289–300.
[33] Shamkant B. Navathe and Rafi Ahmed, *Temporal Relational Model and a Query Language*. Information Sciences, 1989. **49**(1): 147–175.
[34] Benjamin Nguyen, Serge Abiteboul, Grégory Cobena. and Mihaï Preda, "Monitoring XML Data on the Web," in *SIGMOD*. 2001. Santa Barbara, CA, pp. 437–448.
[35] OMG, Unified Modeling Language (UML), v1.5. 2003.

[36] Yannis Papakonstantinou and Victor Vianu, "Incremental Validation of XML Documents," in *ICDT*, 2003. Siena, Italy, pp. 47–63.

[37] Sudha Ram and Jun Liu, "Understanding the Semantics of Data Provenance to Support Active Conceptual Modeling," in *ACM-L Workshop* (*ER*), 2006. Tucson, AZ, pp. 1–12.

[38] John F. Roddick, *A Survey of Schema Versioning Issues for Database Systems*. Information and Software Technology, 1995. **37**(7): 383–393.

[39] John F. Roddick and Denise de Vries, "Reduce, Reuse, Recycle: Practical Approaches to Schema Integration, Evolution and Versioning," in *ER* (*Workshops*), 2006. Tucson, AZ, pp 209–216.

[40] *Service Data Objects for Java Specification, Version 2.01*. `http://www128.ibm.com/developerworks/webservices/-library/specification/ws-sdo`, current as of March 2006.

[41] Jennifer Widom, "Trio: A System for Integrated Management of Data Accuracy, and Lineage," in *CIDR*, 2005. pp. 262–276.

[42] Xyleme, *A Dynamic Warehouse for XML Data of the Web*. IEEE Data Engineering Bulletin, 2001. **24**(2):40–47.

[43] Cong Yu and Lucian Popa, "Semantic Adaptation of Schema Mappings when Schemas Evolve," in *VLDB*, 2005. Trondheim, Norway, pp. 1006–1017.

[44] Shuohao Zhang, Curtis E. Dyreson, and Richard T. Snodgrass. "Schema-Less, Semantics-Based Change Detection for XML Document*s*," in *WISE*, 2004. Brisbane, pp. 279–290.