AN APPROACH FOR SCHEMA VERSIONING IN MULTI-TEMPORAL XML DATABASES

Zouhaier Brahmia and Rafik Bouaziz
Faculty of Economic Sciences and Management, University of Sfax, Road of the Aerodrome, Sfax, Tunisia

Keywords: Temporal databases, XML databases, Schema versioning, XML Schema.

Abstract: Schema evolution keeps only the current data and the schema version after applying schema changes. On the contrary, schema versioning creates new schema versions and preserves old schema versions and their corresponding data. These two techniques have been investigated widely, both in the context of static and temporal databases. With the growing interest in XML and temporal XML data as well as the mechanisms for holding such data, the XML context within which data items are formatted also becomes an issue. Whereas much research work has recently focused on the problem of schema evolution in XML databases, less attention has been devoted to schema versioning in such databases. In this paper, we propose an approach for schema versioning in multi-temporal XML databases. This approach is based on the XML Schema language for describing XML schema, and is database consistency-preserving.

1 INTRODUCTION

XML (W3C, 2006a) is an emergent standard for web documents. It is being used in a wide range of applications and among a wide array of communities. In database context, XML is also a new database model for semi-structured data.

XML provides an excellent support for temporally grouped data models, which have long been advocated as the most natural and effective representations of temporal information (Clifford et al, 1995).

Since change is a fundamental aspect of persistent information and data-centric systems, both the data and the structure (schema) of XML documents tend to change over time for a multitude of reasons, including to reflect a change in the real world, a change in the user's requirements, mistakes in the initial design or to allow the expansion of the application scope over time. While XML schema changes are inevitable during the life of an XML database, most of the current XML DBMS unfortunately do not provide enough support for these changes and do not support schema evolution or schema versioning. So, XML database developers try to solve the problem of schema evolution in an ad hoc manner. Note that schema evolution is partially supported by some relational database systems, such as Oracle and Microsoft SQL Server, and also by some object-oriented database systems, such as Orion and TIGUKAT (Ozsu et al, 1995).

Schema versioning has been previously studied in the context of temporal databases (Roddick, 1995). But an XML schema is a grammar specification, unlike a relational database schema, so new techniques are required. Though various XML schema languages have been proposed in the literature and in the commercial area (Lee & Chu, 2000), there are neither model schema changes nor provision for versioning.

Schema versioning and its consequences on instances have also been thoroughly investigated in object-oriented databases (e.g. (Galante et al, 2005)). Though object-oriented schema bring some similarities with XML Schema, there are fundamental differences that prevent to smoothly adapt techniques developed in that context to XML Schema.

Moreover, whereas several previous works have dealt with schema evolution in XML databases (Coox, 2003; Guerrini et al, 2005; Wang & Zaniolo, 2005), few works were done on schema versioning in such environment (Costello & Utzinger, 2006; W3C, 2006b). Besides, whereas most previous works about XML schema versioning use the DTD schema language (W3C, 2006a) which has limited capabilities compared to other schema languages (Lee & Chu, 2000), few works have used the XML

Brahmia Z. and Bouaziz R. (2008).
AN APPROACH FOR SCHEMA VERSIONING IN MULTI-TEMPORAL XML DATABASES.
In *Proceedings of the Tenth International Conference on Enterprise Information Systems - DISI*, pages 290-297
DOI: 10.5220/0001707702900297

Schema language (W3C, 2001) which is a powerful schema language backed by the W3C and supports the major features available in other XML schema languages. For these reasons, we propose in this paper an approach for schema versioning in multi-temporal XML databases, using the XML Schema language.

The rest of the paper is organized as follows. Section 2 presents our approach for schema versioning in multi-temporal XML databases (chosen XML schema language; schema change operations; schema change propagation; XML data management and querying; implementation). Section 3 is devoted to a brief review of related works. Conclusions can finally be found in Section 4.

2 PRESENTATION OF THE PROPOSED APPROACH

A general description of our approach is depicted in figure 1.

The designer defines, modifies and deletes XML Schema of objects of the real world. These XML schema can evolve over time through many versions. Figure 1 shows the translation from the XML Schema version number i of an object o to the XML Schema version number i+1 of the same object o. After this translation, the version number i becomes a past version and the version number i+1 becomes the current XML Schema version of the object o.

Users must define documents which are valid to a specified XML Schema version (a past version or the current version). By default, users must use the current XML Schema version of an object to define and to modify documents which are valid to this XML Schema version. But a user can specify that he/she wants to define an XML document of an object according to a past XML Schema version of this object. The changes of an XML document must be done within the same XML Schema version of this document, i.e. this document must be valid to its schema even after modification.

2.1 Chosen XML Schema Formalism: XML Schema

Although there are many schema languages (DTD, XML Schema, XDR, DSD, SOX, Schematron, DCD, DDML, RELAX, Assertion Grammars, etc.) (Lee & Chu, 2000), the two mostly widely-used schema languages are currently DTD and XML Schema.

We choose XML Schema (and not DTD) for the following reasons:

- it includes more powerful features for defining the structure and content of an XML document than DTD (it has more than 44 built-in data types available, over only 10 data types for DTD language; it supports different keys like primary key and referenced key as opposed to only ID and IDREF support in DTD, etc.);
- it supports the major features available in other XML schema languages;
- it is backed by the W3C.

The XML Schema language is also referred to as XML Schema Definition (XSD).

2.2 Basic Principles

Our research is based on the following basic principles:

- The considered XML environment is multi-version and temporal: each object of the real world (e.g. Student, Course, Teacher and Degree, in an academic system) can have many XML Schema versions since an XML

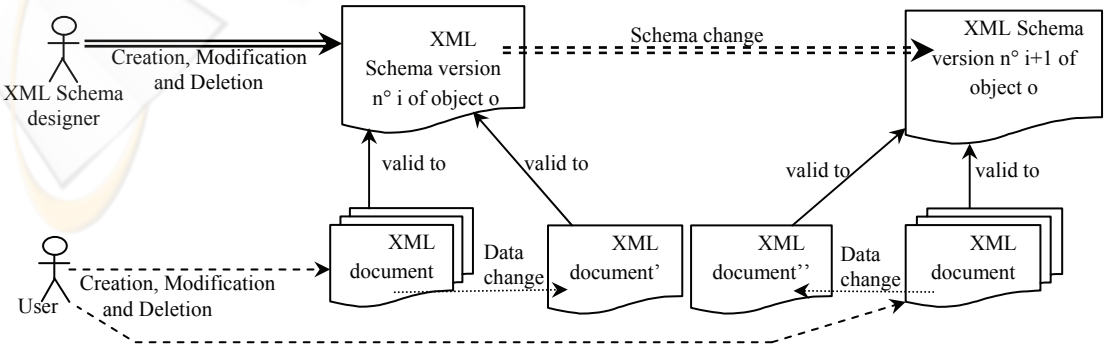

Figure 1: General description of our approach.

Schema of an object is considered as a logical entity that evolves over time through various versions; and each XML Schema version has a temporal format (snapshot, transaction-time, valid-time or bitemporal). An XML document which is valid to an XML Schema version must have the same temporal format of its schema.

- Extensional XML data (or XML documents) are multi-temporal (De Castro et al, 1997): we consider an XML database in which documents of different temporal formats coexist. A temporal XML document records the entire history of an object rather than just its current state.
- XML schema versioning is temporal: any XML schema version has a temporal interval [version start time – version end time]. The version end time remains unknown until the definition of a new XML Schema version; during this period, it is set to the special symbol 'UC' that means 'until changed'.
- XML schema versioning is individual: each object has its own XML Schema versions which are managed independently from XML Schema versions of any other object.
- For each object of the real world, we have always one current XML schema version and zero, one or several past XML schema version(s). When a new XML schema version is defined for an object, this version becomes the only current version for this object and the previous XML Schema version becomes a past version.
- XML Schema versions of an object are numbered in an increasing order (e.g., Employee_V1.xsd, Employee_V2.xsd, Employee_V3.xsd, etc.). The current XML schema version has always the biggest number.
- An XML Schema version can be derived from a previous one: XML database administrator can use old schema versions to define new ones.
- The new XML Schema version must be different from the last one (difference in structure and/or difference in format); otherwise it cannot be accepted. We must not have two successive schema versions which are identical. If we suppose that a schema version V2 is identical to V1, what is the utility of V2 in that case? V2 is not necessary; the DBMS must refuse it and continue considering V1 as the current XML schema version until having a new schema version which modifies V1.

2.3 Schema Change Operations for XML Schema

In this section, we propose a set of basic schema change operations that can be applied to any XML schema expressed by XML Schema language. We also provide examples of their use. These operations provide schema versioning facilities and are consistency-preserving: they ensure that all existing XML Schemas and their underlying XML documents are kept without any modification.

An XML Schema of an object is simply an XML document which is a set of XML elements. Each element can be composed of several sub-elements and each element or sub-element is characterized by a set of attributes. Thus, schema change operations can deal with XML Schema elements (or XML Schema sub-elements) and/or attributes of XML Schema elements (or XML Schema sub-elements).

Translation from the current XML Schema version of an object to a new XML Schema version of this object is achieved by applying a set of schema change operations.

Below, we present an example to illustrate changes in XML Schema. Figure 2 depicts the first version of an XML Schema for the object Hotel (in a system of hotel management), called Guide_V1.xsd, and Figure 3 shows a sample XML document valid to this XML Schema version. The first sample document is used for running examples hence forth.

```
<?xml version=”1.0” encoding=”UTF-8”?>
  <xsd:schema xmlns:xsd=http://www.w3.org/2000/10/XMLSchema>
    <xsd:element name="Guide">
     <xsd:complexTtype>
      <xsd:sequence>
       <xsd:element name="Hotel" maxOccurs="unbounded">
        <xsd:complexType>
         <xsd:sequence>
          <xsd:element name="Name" type="xsd:string"/>
           <xsd:element name="Address">
            <xsd:complexType>
             <xsd:sequence>
              <xsd:element name="Number" type="xsd:short"/>
               <xsd:element name="Street" type=”xsd:string”/>
               <xsd:element name="Town" type=”xsd:string”/>
               <xsd:element name="Postcode" type=”xsd:string”/>
              </xsd:sequence>
             </xsd:complexType>
            </xsd:element>
            <xsd:element name="Room" type="xsd:short"/>
            <xsd:element name="Phone" type="xsd:string"/>
          <xsd:element name="Fax" type="xsd:string" minOccurs=”0”/>
         </xsd:sequence>
      <xsd:attribute name=”Category” type="xsd:string" use=”required”/>
        </xsd:complexType>
       </xsd:element>
      </xsd:sequence>
      <xsd:attribute name=”Country” type="xsd:string" use=”required”/>
     <xsd:attribute name=”Continent” type="xsd:string" use=”required”/>
      <xsd:attribute name=”Version” type="xsd:short" use=”required”/>
```

```
      <xsd:attribute name=”Version_Start_Time” type="xsd:date"
use=”required”/>
       <xsd:attribute name=”Version_End_Time” type="xsd:date"
use=”required”/>
      <xsd:attribute name=”Format” type="xsd:string" use=”required”
fixed=”Snapshot”/>
      </xsd:complexType>
   </xsd:element>
</xsd:schema>
```

Figure 2: Guide_V1.xsd.

```
<?xml version=”1.0” encoding=”UTF-8”?>
<Guide Country=”Tunisia” Continent=” Africa” Version=”1”
   Version_Start_Time=”2007-01-31” Version_Start_Time=”2007-01-31”
   Format=”Snapshot”>
    <Hotel Category=”****”>
        <Name>Kantawi Club</Name>
        <Address>
            < Number>12</Number>
            <Street>Flowers</Street>
            <Town>Sousse</Town>
            <Postcode>3063</Postcode>
        </Address>
        <Room>1200</Room>
        <Phone>+216 73 234 123</Phone>
        <Fax>+216 73 534 123</Fax>
    </Hotel>
  </Guide>
```

Figure 3: An XML document valid to Guide_V1.xsd.

2.3.1 Notations

When dealing with the schema change operations, we will use the notation presented in Table 1.

Table 1: Notation of the schema change operations.

Expression	Description
$SV_{i,o}$	The XML Schema version number i of the object o
$e_{i,o}$	The element e of $SV_{i,o}$
$E_{i,o}$	The set of elements of $SV_{i,o}$; $E_{i,o} = \{e_{i,o}\}$
$a_{i,o}$	The attribute a of $e_{i,o}$
$Att(e_{i,o})$	The set of attributes of the element $e_{i,o}$; $Att(e_{i,o}) = \{a_{i,o}\}$

2.3.2 Schema Change Operation Specification

We distinguish six main schema change operations:

- Addition, Deletion and Modification of an XML Schema element;
- Addition, Deletion and Modification of an attribute of an XML Schema element.

We assume that the following schema change operations are applied to $SV_{i,o}$ and produce $SV_{i+1,o}$ as output.

For the illustrative examples, suppose that we translate from Guide_V1.xsd to Guide_V2.xsd by applying a set of schema change operations.

Addition of a new XML Schema Element “e”. This operation cannot be performed if there is an existing element with the same name of e in the current XML Schema version. Such an operation can be formalized as follows:

$$AddElem(SV_{i,o},e) \rightarrow SV_{i+1,o} \text{ such that } E_{i+1,o} = E_{i,o} \cup \{e\}$$

Example 1: Result of the operation “Addition of the element homepage”.

Before addition: Guide_V1.xsd	After addition: Guide_V2.xsd
……	……
<xsd:element name="Fax"	<xsd:element
type="xsd:string"	name="Fax"
minOccurs=”0”/>	type="xsd:string"
</xsd:sequence>	minOccurs=”0”/>
……	**<xsd:element name="homepage" type="xsd:string" minOccurrs="0" maxOccurrs="1"/>**
	</xsd:sequence>
	……

Deletion of an XML Schema Element “e”. The element e must already exist in the current XML Schema version, otherwise this operation fails. Such an operation can be formalized as follows:

$$DelElem(SV_{i,o},e) \rightarrow SV_{i+1,o} \text{ such that } E_{i+1,o} = E_{i,o} / \{e\}$$

Example 2: Result of the operation “Deletion of the element Fax”.

Before deletion: Guide_V1.xsd	After deletion: Guide_V2.xsd
……	……
<xsd:element name="Phone"	<xsd:element
type="xsd:string"/>	name="Phone"
<xsd:element name="Fax" type="xsd:string" minOccurs=”0”/>	type="xsd:string"/>
</xsd:sequence>	</xsd:sequence>
……	……

Modification of an XML Schema Element “e”. The element e must exist in the current XML schema version, otherwise this operation cannot be realized. This operation can be one of the following operations: Addition of a new sub-element s to e, Deletion of a sub-element s of e, Renaming e, Modification of type of element e, etc. It can be formalized as follows:

$$ModifElem(SV_{i,o},e) \rightarrow SV_{i+1,o} \text{ such that } E_{i+1,o} = E_{i,o} / \{e_{i,o}\} \cup \{e_{i+1,o}\}$$

Example 3: Result of the operation "Modification of the element Postcode".

Before modification: Guide_V1.xsd	After modification: Guide_V2.xsd
……	……
<xsd:element	<xsd:element
name="Town"	name="Town"
type="xsd:string"/>	type="xsd:string"/>
<xsd:element	<xsd:element
name="Postcode"	name="**Zip code**"
type="xsd:string"/>	type="xsd:string"/>
</xsd:sequence>	</xsd:sequence>
……	……

Addition of an Attribute "a" to an XML Schema Element "e". This operation cannot be achieved if there is an existing attribute of e with the same name of a, in the current XML Schema version. It can be formalized as follows:

$$AddAtt(SV_{i,o},e,a) \rightarrow SV_{i+1,o} \text{ such that } E_{i+1,o} = E_{i,o} / \{e_{i,o}\} \cup \{e_{i+1,o}\} \wedge [Att(e_{i+1,o})=Att(e_{i,o}) \cup \{a_{i+1,o}\}]$$

Example 4: Result of the operation "Addition of the attribute Id to the element Hotel".

Before addition: Guide_V1.xsd	After addition: Guide_V2.xsd
……	……
</xsd:sequence>	</xsd:sequence>
<xsd:attribute	**<xsd:attribute**
name="Category"	**name="Id"**
type="xsd:string"	**type="xsd:byte"**
use="required"/>	**use="required"/>**
</xsd:complexType>	<xsd:attribute
……	name="Category"
	type="xsd:string"
	use="required"/>
	</xsd:complexType>
	……

Deletion of an Attribute "a" of an XML Schema Element "e". This operation cannot be performed if there is no existing attribute of e with the same name of a, in the current XML Schema version. Its formalization is as follows:

$$DelAtt(SV_{i,o},e,a) \rightarrow SV_{i+1,o} \text{ such that } E_{i+1,o} = E_{i,o} / \{e_{i,o}\} \cup \{e_{i+1,o}\} \wedge [Att(e_{i+1,o})=Att(e_{i,o}) / \{a_{i,o}\}]$$

Example 5: Result of the operation "Deletion of the attribute Continent of the element Guide".

Before deletion: Guide_V1.xsd	After deletion: Guide_V2.xsd
……	……
<xsd:attribute	<xsd:attribute
name="Country"	name="Country"
type="xsd:string"	type="xsd:string"
use="required"/>	use="required"/>
<xsd:attribute	<xsd:attribute
name="Continent"	name="Version"
type="xsd:string"	type="xsd:short"
use="required"/>	use="required"/>
<xsd:attribute	……
name="Version"	
type="xsd:short"	
use="required"/>	
……	

Modification of an Attribute "a" of an XML Schema Element "e". This operation cannot be performed if there is no existing attribute of e with the same name of a, in the current XML Schema version. This operation can modify any attribute a of e (like its name or its type). It can be formalized as follows:

$$ModifAtt(SV_{i,o},e,a) \rightarrow SV_{i+1,o} \text{ such that } E_{i+1,o} = E_{i,o} / \{e_{i,o}\} \cup \{e_{i+1,o}\} \wedge [Att(e_{i+1,o})=Att(e_{i,o}) / \{a_{i,o}\} \cup \{a_{i+1,o}\}]$$

Example 6: Result of the operation "Modification of the attributes Version_Start_Time and Version_End_Time of the element Guide".

Before modification: Guide_V1.xsd	After modification: Guide_V2.xsd
……	……
<xsd:attribute	<xsd:attribute
name="Version_Start_Time" type="xsd:date"	name="Version_Start_Time" type="**xsd:dateTime**"
use="required"/>	use="required"/>
<xsd:attribute	<xsd:attribute
name="Version_End_Time"	name="Version_End_Time"
type="xsd:date"	type="**xsd:dateTime**"
use="required"/>	use="required"/>
……	……

2.4 Schema Change Propagation

In order to avoid problems, like data loss, modification of semantics and essentially document revalidation (Raghavachari & Shmueli, 2004; Guerrini et al, 2005), and to preserve the complete history of XML Schema and data changes, creation of a new XML schema version must not affect old XML schema versions and their underlying XML documents. Old XML documents continue to be considered as valid for their XML Schema and must not be revalidated. But new XML documents should be valid for the new schema version. In our approach, when a new XML Schema version is

created, there is neither conversion of previous XML schema nor revalidation of previous XML documents. Only the version end time attribute of the previous XML Schema version, which is set to 'UC', will be changed to the instant that precedes the start time of the new version.

Thus, schema changes do not affect the underlying data.

Example: Suppose we have an XML Schema version V_i ($i > 0$) used during its temporal interval $[t_i - UC[$. When we apply to V_i, at an instant t_j ($j > i$), one or more schema change operations (addition of an element, for example), a new XML schema version V_{i+1} will be created by the XML DBMS with a temporal interval equal to $[t_j - UC[$. The old XML schema version V_i will be kept with a temporal interval equal to $[t_i - (t_j - 1)]$. All documents which are valid relatively to V_i will also be kept without any modification.

2.5 XML Data Management

In our context, XML documents are multi-version and temporal. XML data management includes XML document creation, modification and deletion. XML document modification includes XML element addition, modification and deletion. In an XML document of snapshot or valid-time format, XML element modification and deletion operations are destructive. However, these operations are not destructive in an XML document of transaction-time or bitemporal format. The definition of each operation is presented below.

Addition of an Element. This operation inserts a new element into the corresponding position in the XML document. If this XML document is of transaction-time or a bitemporal format, then the system sets the transaction start time attribute of this new element to the current time and the transaction end time attribute to the special symbol 'UC'. In the case of a valid-time or a bitemporal document, the validity start time and validity end time attributes of the new element are defined by the user. The validity end time attribute can be set by the user to the special symbol 'now' that represents the ever-increasing current element.

Modification of an Element. This operation appends a new element with the same name immediately after the original element. If the corresponding XML document is of transaction-time or a bitemporal format, then the system sets the transaction start time attribute of this new element to the current time and the transaction end time attribute to 'UC'; it also sets the transaction end time attribute of the original element to the current time.

Each previous XML document, which is valid to a previous XML schema version, must be modified according to its schema version.

After modification of an XML document, this latter must be valid to its original XML Schema, else the modification is canceled and the document is kept in its initial state (i.e. before modification).

Deletion of an Element. This operation changes the state attribute of this element to 'd' that means 'deleted'. This attribute can be set to 'v' (default value) for 'valid' or 'e' for 'erroneous'.

For each object of the real world, new XML documents are created and modified according to the current XML schema version of this object; they must be valid to this version.

Thus, XML data (or document) change operations (element addition, modification or deletion) have no effect on the corresponding XML Schema. Modification of an XML document cannot lead to a new XML Schema; it has to leave the modified document valid with regard to its XML Schema.

2.6 XML Data Querying

In a multi-version environment, we can have multi-schema queries (i.e., queries in which several schema versions qualify for the temporal selection conditions). To answer these queries, the XML DBMS will use all selected schema versions (multi-schema answer).

XQuery (W3C, 2007) is a powerful query language which provides an effective support for (complex) temporal queries (e.g. slicing queries, historical queries and temporal join queries) at the logical level (Wang & Zaniolo, 2005).

In order to support temporal multi-schema queries, XQuery will need new operators like the Allen operators (Allen, 1983) (i.e. before, after, meets, overlaps, etc.) and new functions in order to specify schema versions in XML queries (e.g. first_verion(), last_version(), version_applied_at(a particular time t), versions_applied_during(a particular time interval), etc.).

XQuery is Turing-complete and natively extensible (Kepser, 2004). Thus, additional constructs needed for temporal multi-schema queries can be defined in XQuery itself, without having to depend on difficult-to-obtain extensions by standard committees.

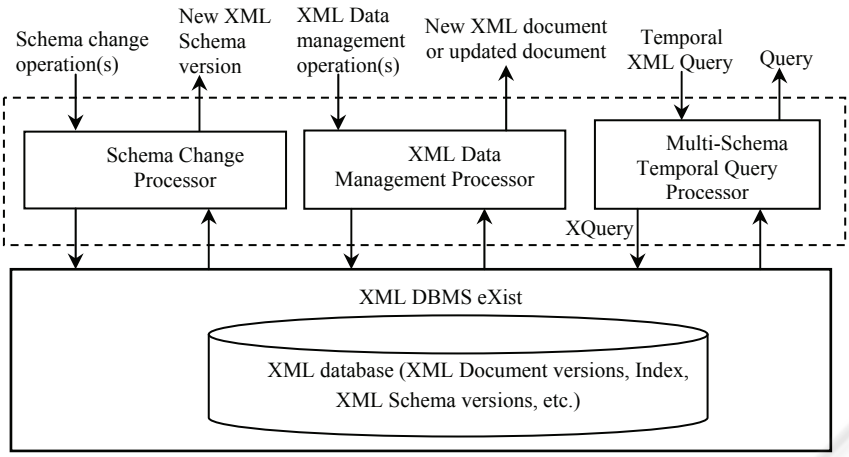

Figure 4: The overall system architecture.

2.7 Implementation

To verify the feasibility of our approach, we implement the ideas presented in this paper in a functioning prototype **sy**stem for **s**chema **ver**sioning in **mu**lti-**te**mporal **X**ML databases, called Sysvermutex. This system is developed on top of an open source native XML DBMS, eXist (http://exist-db.org/), by using Java, within a stratum approach. The overall architecture is depicted in figure 4. The dashed rectangle indicates the boundary of the stratum. This stratum consists of the three components shown as Schema Change Processor, XML Data Management Processor and Multi-Schema Temporal Query Processor modules.

3 RELATED WORK

Schema evolution had been previously investigated for schemas expressed by DTDs in (Su et al, 2001), where a set of evolution operators is proposed and discussed in detail. DTD evolution has also been investigated in (Bertino et al, 2002): the focus was on dynamically adapting the schema to the structure of most documents stored in an XML data source. An axiomatic model of an XML database schema is suggested in (Coox, 2003) that automatically maintains the integrity of the XML database when basic changes are made to its schema. A framework for schema evolution of relational database-based systems with XML "interface" is presented in (Simanovsky, 2004).

Schema evolution and schema versioning had also been studied for schema expressed by XML Schema in (Guerrini et al, 2005), (Dyreson et al, 2006) and (Joshi, 2007). In (Guerrini et al, 2005), the authors have proposed a set of evolution primitives and analyzed the impact of such primitives on the validity of XML documents known to be valid for the original schema. In (Dyreson et al, 2006), the authors show how schema versioning can be integrated with support for time-varying documents in a fashion consistent and upwardly-compatible with XML, XML Schema, and conventional XML validators. In (Joshi, 2007), the authors show that by utilizing schema-constant periods and cross-wall validation, it is possible to realize a comprehensive system for representing and validating data- and schema-versioned XML documents, while remaining fully compatible with the XML standards.

Moreover, in order to minimize impact to existing instance documents and applications as new versions of XML Schema are created and to facilitate system evolution, the authors of (Costello & Utzinger, 2006) made eight recommendations concerning design of XML Schema, instance documents and applications. A description of use cases where XML Schema are being versioned is presented in (W3C, 2006b). These use cases describe the desired behaviour from XML Schema processors when they encounter the different versions of schema and the instances defined by them.

Finally, it is shown in (Wang & Zaniolo, 2005) that XML views combined with XML query languages (like XQuery) can provide surprisingly effective solutions to the problem of representing and querying both the evolution of database contents and the evolution of database schema.

4 CONCLUSIONS

This paper presents a new approach for schema versioning in multi-temporal XML databases.

This approach is based on XML Schema which is a powerful XML schema language that overcomes the limitations of DTDs (used in most works concerning schema evolution or versioning).

Moreover, our approach treats XML Schema changes as a versioning process instead of a simple evolution. It also ensures the consistency of the XML database since:

- when a new XML Schema version is defined, it does not convert previous XML Schema versions and does not revalidate previous XML documents which are valid to their XML Schema versions;
- XML document change operations do not affect the corresponding XML Schema;
- after modification of an XML document, this latter is still valid to its XML Schema.

The prototype implementation of the proposed approach will serve as a testbed for experimental evaluation.

Currently, we are working on XML data change operations in a temporal multi-version environment and on extensions needed by XQuery to support temporal multi-schema queries.

REFERENCES

Allen, J.F., 1983. Maintaining knowledge about temporal intervals. *Communications of the ACM,* 26(11), p.832-843.

Bertino, E., Guerrini, G., Mesiti, M. & Tosetto, L., 2002. Evolving a Set of DTDs according to a Dynamic Set of XML Documents. In *EDBT Workshops 2002, 8th International Conference on Extending Database Technology 2002 Workshops*. Springer.

Clifford, J., Croker, A., Grandi, F. & Tuzhilin, A., 1995. On Temporal Grouping. In *Temporal Databases 1995, International Workshop on Temporal Databases 1995*. Springer.

Coox, S.V., 2003. Axiomatization of the Evolution of XML Database Schema. *Programming and Computer Software,* 29(3), p.1-7.

Costello, R. L. & Utzinger, M., 2006. Impact of XML Schema Versioning on System Design. www.xfront.com/SchemaVersioning.html

De Castro, C., Grandi, F. & Scalas, M.R., 1997. Schema versioning for multitemporal relational databases. *Information Systems,* 22(5), p.249-290.

Dyreson, C., Snodgrass, R. T., Currim, F., Currim, S. & Joshi, S., 2006. Validating Quicksand: Schema Versioning in τXSchema. In *ICDE Workshops 2006, 22nd International Conference on Data Engineering Workshops*. IEEE Computer Society.

Galante, R.M., Dos Santos, C.S., Edelweiss, N. & Moreira, A.F., 2005. Temporal and versioning model for schema evolution in object-oriented databases. *Data and Knowledge Engineering,* 53(2), p.99-128.

Guerrini, G., Mesiti, M. & Rossi, D., 2005. Impact of XML Schema Evolution on Valid Documents. In *WIDM'05, 7th ACM International Workshop on Web Information and Data Management*. ACM.

Joshi, S., 2007. τXSchema - Support for Data- and Schema-Versioned XML Documents. Technical Report TR-89, TimeCenter. http://www.cs.auc.dk/TimeCenter/

Kepser, S., 2004. A Simple Proof for the Turing-Completeness of XSLT and XQuery. In *EML2004, Extreme Markup Languages 2004 Conference*. IDEAlliance.

Lee, D. & Chu, W.W., 2000. Comparative Analysis of Six XML Schema Languages. *ACM SIGMOD Record,* 29(3), p.76-87.

Ozsu, M.T., Peters, R.J., Szafron, D., Irani, B., Lipka, A. & Munõz, A., 1995. TIGUKAT: a uniform behavioral objectbase management system. *The VLDB Journal,* 4(3), p.445-492.

Raghavachari, M. & Shmueli, O., 2004. Efficient Schema-Based Revalidation of XML. In *EDBT 2004, 9th International Conference on Extending Database Technology*. Springer.

Roddick, J., 1995. A survey of schema versioning issues for database systems. *Information and Software Technology,* 37(7), p.383-393.

Simanovsky, A., 2004. Evolution of Schema of XML-documents Stored in a Relational Database. In *DB&IS'2004, 6th International Baltic Conference on Database and Information Systems*. Springer-Verlag.

Su, H., Kramer, D., Chen, L., Claypool, K. T. & Rundensteiner, E. A., 2001. XEM: Managing the evolution of XML documents. In *RIDE-DM'01, 11th International Workshop on Research Issues in Data Engineering: Document Management for Data Intensive Business and Scientific Applications*. IEEE Computer Society.

Wang, F. & Zaniolo, C., 2005. An XML-Based Approach to Publishing and Querying the History of Databases. *World Wide Web,* 8(3), p.233-259.

W3C, 2001. XML Schema Part 0: Primer. W3C Recommendation. http://www.w3.org/TR/2001/REC-xmlschema-0-20010502/

W3C, 2006a. Extensible Markup Language (XML) 1.0 (4th edition). W3C Recommendation. http://www.w3.org/TR/2006/REC-xml-20060816.

W3C, 2006b. XML Schema Versioning Use Cases. W3C Working Draft. http://www.w3.org/XML/2005/xsd-versioning-use-cases/2006-01-31.html.

W3C, 2007. XQuery 1.0: An XML Query Language. W3C Recommendation. http://www.w3.org/TR/2007/REC-xquery-20070123/