A Study of Conventional Schema Versioning in the τXSchema Framework

Zouhaier Brahmia, Rafik Bouaziz, Fabio Grandi, Barbara Oliboni

June 13, 2012

TR-94

A TIMECENTER Technical Report

Title	A Study of Conventional Schema Versioning in the τXSchema Framework
Author(s)	Zouhaier Brahmia, Rafik Bouaziz, Fabio Grandi, Barbara Oliboni
Publication History	June 2012. A TIMECENTER Technical Report.

The TIMECENTER icon on the cover combines two "arrows." These "arrows" are letters in the so-called *Rune* alphabet used one millennium ago by the Vikings, as well as by their precedessors and successors. The Rune alphabet (second phase) has 16 letters, all of which have angular shapes and lack horizontal lines because the primary storage medium was wood. Runes may also be found on jewelry, tools, and weapons and were perceived by many as having magic, hidden powers.

The two Rune arrows in the icon denote "T" and "C," respectively.

A Study of Conventional Schema Versioning in the τXSchema Framework

Zouhaier Brahmia, Rafik Bouaziz
University of Sfax, Tunisia − email: {zouhaier.brahmia|raf.bouaziz}@fsegs.rnu.tn
Fabio Grandi
University of Bologna, Italy − email: fabio.grandi@unibo.it
Barbara Oliboni
University of Verona, Italy − email: barbara.oliboni@univr.it

Abstract

Schema versioning is an indispensable feature for applications using temporal databases and requiring an entire history of data and schema. τXSchema [6] is an infrastructure for constructing and validating temporal XML documents; but any explicit support for XML schema versioning is offered. A τXSchema schema is composed of a conventional XML Schema document annotated with physical and logical annotations. All components of a τXSchema schema (i.e., conventional schema, logical annotations, and physical annotations) can change over time to reflect changes in user requirements or in reference world of the database. In a previous work [9], we deal with versioning of logical and physical annotations. In this work, we study τXSchema conventional schema versioning: we propose a complete set of low-level primitives for changing such a schema and define their operational semantics.

Keywords: τXSchema, Schema versioning, XML, XML Schema, Temporal database

1 Introduction

Nowadays, lots of applications use XML repositories to store information. Some examples include e-government, electronic commerce, internet marketing, electronic data interchange, and electronic funds transfer. In such repositories, we find XML documents which contain data and XML schema which describe structures of these documents. Furthermore, both data and schema evolve over time to meet changes in universe of discourse or user requirements; data and schema updates are unavoidable in any information system.

In this context, several applications using XML repositories (e.g., banking, accounting, personnel management, airline reservations, weather monitoring and forecasting) are temporal in nature and require a full history of data and schema changes, which must be managed efficiently, consistently, and in a transparent way with regard to the end user. Notice that for generic temporal databases [1], XML provides an excellent support for temporally grouped data models [2], which have long been considered as the most natural and effective representations of temporal information [3]. Besides, schema versioning has long been advocated to be the more appropriate solution to support a complete data and schema history in databases [4,5].

In a temporal setting, XML data can evolve along transaction-time and/or valid-time; thus, they can have a transaction-time, a valid-time or a bitemporal format. When XML data of different temporal formats can coexist in the same XML repository, we talk about a multitemporal XML repository. Whereas schema versioning is required by several applications using multitemporal XML repositories, both existing XML DBMS and XML tools have no support for that feature until now. Therefore, XML Schema designers and developers use ad hoc methods to manage schema versioning. To reach our goal which consists of proposing

a general approach for schema versioning in multitemporal XML repositories, we have chosen to extend the τXSchema approach [6,7,8] for reasons presented in [9]. τXSchema [6,7,8] is a framework (a language and a suite of tools) for the creation and validation of time-varying XML documents. The τXSchema language extends the standard XML Schema language [10] to explicitly support time in the definition of temporal XML documents. In τXSchema, the schema is obtained by specifying three parts: (i) the conventional schema which is a standard XML Schema document that describes the structure of a standard XML document, without any temporal aspect; (ii) the logical annotations of the conventional schema, which identify which elements can vary over time; and (iii) the physical annotations of the conventional schema, which describe how the time-varying aspects are represented.

In the first works on τXSchema [6,7,8], the authors focus on capturing a time-varying XML schema and validating XML documents against such a schema. In [6], they introduce τXSchema but did not discuss schema versioning. In [7], they focus on cross-schema change validation. In [8], they extend [7] by discussing how to accommodate gaps in the existence time of an item, transaction semantics, and non-sequenced integrity constraints. In all these works, they do not deal with how the schema changes are made, or what kinds of schema change operations are supported. We have started studying these issues in our last work [9] on τXSchema: we dealt with changes of physical and logical annotations. We proposed a set of low-level primitives acting on the annotation document. When the designer decides to make a change on annotations, he/she applies a set of these primitives on the XML file which stores the annotations and he/she gets a new version of the annotations.

In the present work, we study changes within the other component of τXSchema: the conventional schema (or the basic schema). We propose a complete set of primitives allowing the designer to do any change on this schema, by composing these primitives into valid sequences and collectively executing them on the conventional schema. This set of primitives is complete, that is each conventional schema can be generated starting from the empty schema by applying a sequence of primitives, and for each conventional schema a sequence of primitives exists for transforming it in the empty schema. Moreover, this set is sound: i.e., each primitive applied to a consistent conventional schema produces a consistent conventional schema.

The remainder of this paper is organized as follows. We briefly present the τXSchema framework in the next section. In Section 3, we introduce our approach for versioning of τXSchema conventional schema. Since a conventional schema is defined using the standard XML Schema language [10], we describe in Section 4 the different elements that can compose an XML Schema. We propose in Section 5 the set of primitives for changing the conventional schema. In Section 6, we discuss related work. We conclude the work in Section 7.

2. The τXSchema Framework

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

2.1. Architecture

The τXSchema framework [6,7,8,11] allows a designer to create a temporal XML schema for temporal XML documents from a conventional schema (written in standard XML Schema language), logical annotations, and physical annotations. Figure 1 illustrates the architecture of τXSchema [11]. We note that only the components which are shaded in the figure are specific to an individual time-varying document and need to be supplied by a designer.

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

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

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

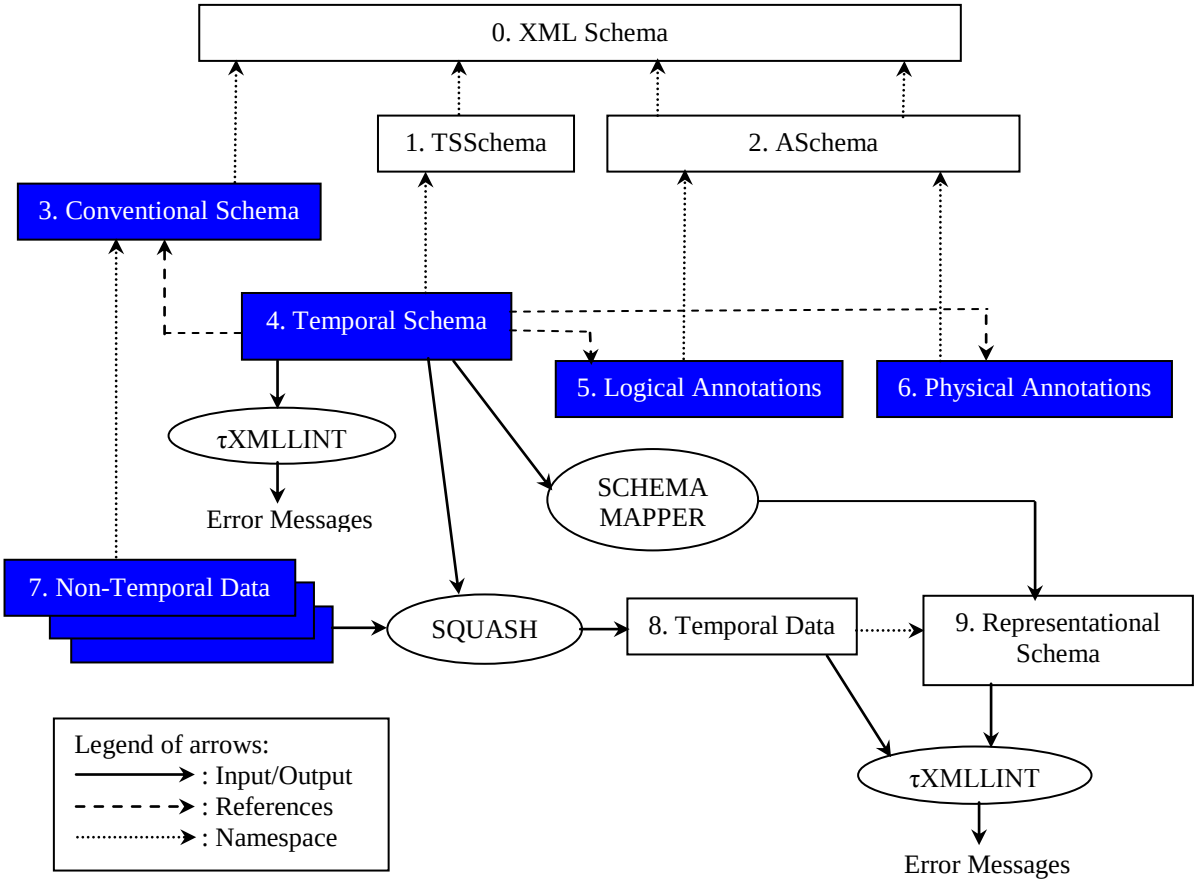

Figure 1. Architecture of τXSchema

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

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

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

Notice that, whereas the introduction of TSSchema (box 1) and ASchema (box 2) is due to Snodgrass and colleagues, XML Schema (box 0) is the standard endorsed by the W3C [10].

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

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

2.2. Motivating example

Let us resume the example presented in [9] (section 2.2), dealing with the management of customer accounts in a bank. In that example, we have one version for the conventional schema (see Figure 2), three versions for the annotation document, and the temporal schema document (see Figure 3), on August 1, 2010.

We do not consider here the annotation document and its versions, since they are beyond the scope of the present work.

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

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

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

Figure 3. Temporal schema (BankTemporalSchema.xml) on August 1, 2010

3. Versioning of τXSchema Conventional Schema

In this section, we first briefly describe how τXSchema conventional schema is versioned in our approach, and then we describe the effects of versioning the conventional schema.

3.1. Conventional schema versioning technique

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

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

Changing the conventional schema leads to a new version of it. Similarly, changing a logical or a physical annotation leads to a new version of the whole annotation document. Therefore, the temporal schema is updated after each change of the conventional schema or of the annotation document. In this paper, we deal with changes of the conventional schema.

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

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

3.2. Effects of versioning the conventional schema

Having different versions of the conventional schema is very important. It allows:

(i) reusing of legacy applications developed against past XML schema and documents;

(ii) supporting of complex temporal queries involving one past XML schema version or many XML schema versions (i.e., multi-schema XML queries);

(iii) keeping an efficient archive of XML schema and data history.

In fact, after defining a new conventional schema version:

- XML documents which were valid for any past conventional schema version continue to be

valid;

- programs (which contain XML queries and XML updates) working on existing conventional schema versions and their associated XML documents remain operational;
- document access policies, such as access control policies or index reorganization, defined for existing XML documents should not be revised.

Notice that conventional schema versioning does not lead automatically to proliferation of schema versions. The creation of a new conventional schema version is anyway a seldom task during the XML repository lifetime, which can only be performed by an administrator or a designer of this repository. This task may consist of dozens of schema change primitives which are grouped together in the same single transaction. But, if application context is very dynamic and fast-growing (i.e., systems that are characterized by a collaborative and a distributed nature of their development and content management, like content management systems, wiki-based web portals, and web systems for large collaborative scientific projects relying on very large scientific databases), schema changes become very frequent and obviously the number of schema versions will increase; it is natural and unavoidable. Whatever is the application context, if the need arises, XML schema have to be changed to conform to real-world changes, to new requirements and constraints, etc.

4. XML Schema Language

XML Schema [10] is a standard language to define the schema for XML documents. It is worth mentioning that we do not address here the full XML Schema definition (e.g. involving the latest W3C Part 1: Structures and Part 2: Datatypes recommendations) which is quite complex, but we focus indeed on a subset of it which we consider very significant for applications, that is the latest W3C Part 0 recommendation [10].

An XML Schema consists of several elements, which describe the structure and the content of any XML document supposed to be valid with respect to such schema. These elements are summarized in Table 1 (their total number is forty-two); for each element, we present its attributes and its containers (i.e., where the element can be included).

Table 1. XML Schema elements

XML Schema element	Attributes of the element	Containers of the element
schema	id, attributeFormDefault, blockDefault, elementFormDefault, finalDefault, targetNamespace, version, and xml:lang	It can not be included in any other XML Schema element
include	id and schemaLocation	schema
import	id, schemaLocation, and namespace	schema
redefine	id and schemaLocation	schema
element	id, name, type, abstract, block, default, final, fixed, form, maxOccurs, minOccurs, nillable, ref, and substitutionGroup	all, choice, schema, or sequence
group	id, name, maxOccurs, minOccurs, and ref	complexType, choice, redefine, schema, or sequence
any	id, namespace, processContents, maxOccurs, and minOccurs	choice or sequence

attribute	id, default, fixed, form, name, ref, type, and use	attributeGroup, complexType, extension, or schema
attributeGroup	id, name, and ref	attributeGroup, complexType, extension, redefine or schema
anyAttribute	id, namespace, and processContents	attributeGroup, complexType, or extension
complexType	id, name, default, final, mixed, abstract, and block	element, redefine, or schema
all	id, minOccurs, and maxOccurs	complexType or group
choice	id, minOccurs, and maxOccurs	choice, sequence, complexType, or group
sequence	id, minOccurs, and maxOccurs	choice, sequence, complexType, or group
complexContent	id and mixed	complexType
simpleContent	id	complexType
extension	id and base	simpleContent or complexContent
simpleType	id, name, and final	attribute, element, redefine, list, union, restriction, or schema
list	id and itemType	simpleType
union	id and memberTypes	simpleType
restriction	id and base	simpleType, simpleContent or complexContent
enumeration	id and value	restriction
pattern	id and value	restriction
totalDigits	id, value, and fixed	restriction
fractionDigits	id, value, and fixed	restriction
minInclusive	id, value, and fixed	restriction
maxInclusive	id, value, and fixed	restriction
minExclusive	id, value, and fixed	restriction
maxExclusive	id, value, and fixed	restriction
minLength	id, value, and fixed	restriction
maxlength	id, value, and fixed	restriction
length	id, value, and fixed	restriction
whiteSpace	id, value, and fixed	restriction
key	id and name	element
keyref	id, name and refer	element
unique	id and name	element
selector	id and xpath	key, keyref, or unique
field	id and xpath	key, keyref, or unique
notation	id, name, public, and system	schema
annotation	id	all, any, anyAttribute, attribute, attributeGroup, choice, complexContent, complexType, element, enumeration, extension, field, fractionDigits, group, import, include, key, keyref, length, list, maxExclusive, maxInclusive, maxLength, minExclusive, minInclusive, minLength, notation, pattern, redefine, restriction, selector, sequence, simpleContent, simpleType, totalDigits, union, unique, whiteSpace, or schema
appinfo	source	annotation
documentation	source and xml:lang	annotation

5. Primitives for Changing τXSchema Conventional Schema

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

change, we describe its arguments and its operational semantics.

5.1. Design choices

The definition of the primitives will obey the following principles and conventions:

a) All primitives must work on a well-formed and valid Conventional Schema (CS), that is must have a well-formed and valid CS as input and produce a well-formed and valid CS as output.

b) All primitives need to work on an XSD file storing the CS, whose name must be supplied as argument.

c) For all primitives, arguments which are used to identify the object on which the primitive works are in the first place of the argument list.

d) Primitives adding elements with possibly optional attributes have the values for all the attributes as arguments; empty places in the argument list stand for unspecified optional attributes.

e) For primitives changing elements (i.e., set primtives), values are specified only for attributes that are changed; the value "unchanged" means that the corresponding attribute is not updated; an empty place in the argument list means that the corresponding attribute receives a nil value.

5.2. Proposed primitives

In this section, we propose the set of primitives for changing a conventional schema (their total number is one hundred and twenty-six). The idea is that each primitive deals with an XML Schema element, that is it adds, deletes or modifies attributes of such an element. For this reason, some arguments of the primitives are the attributes of the corresponding XML Schema elements. We list these primitives in Table 2. Furthermore, due to space limitations, we do not present in this work the effects of all primitive changes. We give only the effect of some of them. In the following, we choose to present only the effects of **`CreateConventionalSchema`**, **`AddInclude`**, **`AddElement`**, and **`AddAny`** primitive changes.

Table 2. Primitive changes acting on the conventional schema

Primitive change	Description
`CreateConventionalSchema(CS.xsd)`	It produces a valid conventional schema (CS) that contains an empty <schema/> element without attributes. According to the design choice (b), the argument is the name of the XSD file where the new CS is stored.
`DropConventionalSchema(CS.xsd)`	It removes the CS.xsd file from disk, with the constraint that the argument represents an empty CS (i.e., like the one above initially created by CreateConventionalSchema). Any other contents must have been removed before.
`SetSchema(CS.xsd, id, targetNamespace, elementFormDefault, attributeFormDefault, blockDefault, finalDefault, version, xml:lang)`	Changes (or introduces) id, targetNamespace, elementFormDefault, attributeFormDefault, blockDefault, finalDefault, version, or xml:lang attributes of the <schema/> element. Notice that the id attribute is optional in any XML Schema component.
`AddInclude(CS.xsd, precedingComponentPath, id, schemaLocation)`	Adds the <include/> element with specified id and schemaLocation to the <schema/> container, after the XML Schema component located at the position precedingComponentPath. If this position is not specified by the designer, it means that the <include/> element is the first element in the <schema/> container.
`DeleteInclude(CS.xsd, includePath)`	Removes the <include/> element located at the position includePath, from the <schema/> container.
`SetInclude(CS.xsd, includePath, id, schemaLocation)`	Changes (or introduces) id or schemaLocation attributes of the <include/> element located at the position includePath, in the <schema/> container.

AddImport(CS.xsd, precedingComponentPath, id, namespace, schemaLocation)	Adds the <import/> element with specified id, namespace, and schemaLocation to the <schema/> container, after the XML Schema component located at the position precedingComponentPath. If this position is not specified by the designer, it means that the <import/> element is the first element in the <schema/> container.
DeleteImport(CS.xsd, importPath)	Removes the <import/> element located at the position importPath, from the <schema/> container.
SetImport(CS.xsd, importPath, id, namespace, schemaLocation)	Changes (or introduces) id, namespace or schemaLocation attributes of the <import/> element located at the position importPath, in the <schema/> container.
AddRedefine(CS.xsd, precedingComponentPath, id, schemaLocation)	Adds the <redefine/> element with specified id and schemaLocation to the <schema/> container, after the XML Schema component located at the position precedingComponentPath. If this position is not specified by the designer, it means that the <redefine/> element is the first element in the <schema/> container.
DeleteRedefine(CS.xsd, redefinePath)	Removes the <redefine/> element located at the position redefinePath, from the <schema/> container.
SetRedefine(CS.xsd, redefinePath, id, schemaLocation)	Changes (or introduces) id or schemaLocation attributes of the <redefine/> element located at the position redefinePath, in the <schema/> container.
AddElement(CS.xsd, toWhat, parentComponentPath, precedingComponentPath, id, name, type, default, fixed, abstract, final, ref, minOccurs, maxOccurs, block, form, nillable, substitutionGroup)	Adds the <element/> element with specified id, name, type, default, fixed, abstract, final, ref, minOccurs, maxOccurs, block, form, nillable, and substitutionGroup to the toWhat (i.e. <all/>, <choice/>, <sequence/>, or <schema/>) container, at the position defined by the path of its parent XML Schema component (parentComponentPath) and the path of its preceding XML Schema component (precedingComponentPath). If this latter is not specified by the designer, it means that the <element/> element is the first element in the toWhat container (or the parent XML Schema component).
DeleteElement(CS.xsd, fromWhat, elementPath)	Removes the <element/> element located at the position elementPath, from the fromWhat (i.e. <all/>, <choice/>, <sequence/>, or <schema/>) container.
SetElement(CS.xsd, inWhat, elementPath, id, name, type, default, fixed, abstract, final, ref, minOccurs, maxOccurs, block, form, nillable, substitutionGroup)	Changes (or introduces) id, name, type, default, fixed, abstract, final, ref, minOccurs, maxOccurs, block, form, nillable, or substitutionGroup attributes of the <element/> element located at the position elementPath, in the inWhat (i.e. <all/>, <choice/>, <sequence/>, or <schema/>) container.
AddGroup(CS.xsd, toWhat, parentComponentPath, precedingComponentPath, id, name, ref, minOccurs, maxOccurs)	Adds the <group/> element with specified id, name, minOccurs, maxOccurs, and ref to the toWhat (i.e. <choice/>, <complexType/>, <redefine/>, <schema/>, or <sequence/>) container, at the position defined by the path of its parent XML Schema component (parentComponentPath) and the path of its preceding XML Schema component (precedingComponentPath). If this latter is not specified by the designer, it means that the <group/> element is the first element in the toWhat container (or the parent XML Schema component).
DeleteGroup(CS.xsd, fromWhat, groupPath)	Removes the <group/> element located at the position groupPath, from the fromWhat (i.e. <choice/>, <complexType/>, <redefine/>, <schema/>, or <sequence/>) container.
SetGroup(CS.xsd, inWhat, groupPath, id, name, ref, minOccurs, maxOccurs)	Changes (or introduces) id, name, ref, minOccurs, or maxOccurs attributes of the <group/> element located at the position groupPath, in the inWhat (i.e. <choice/>, <complexType/>, <redefine/>, <schema/>, or <sequence/>) container.
AddAny(CS.xsd, toWhat, parentComponentPath, precedingComponentPath, id, namespace, minOccurs, maxOccurs, processContents)	Adds the <any/> element with specified id, namespace, minOccurs, maxOccurs, and processContents to the toWhat (i.e. <choice/> or <sequence/>) container, at the position defined by the path of its parent XML Schema component (parentComponentPath) and the path of its preceding XML Schema component (precedingComponentPath). If this latter is not specified by the designer, it means that the <any/> element is the first element in the toWhat container (or the parent XML Schema component).

DeleteAny(CS.xsd, fromWhat, anyPath)	Removes the <any/> element located at the position anyPath, from the fromWhat (i.e. <choice/> or <sequence/>) container.
SetAny(CS.xsd, inWhat, anyPath, id, namespace, minOccurs, maxOccurs, processContents)	Changes (or introduces) id, namespace, minOccurs, maxOccurs, or processContents attributes of the <any/> element located at the position anyPath, in the inWhat (i.e. <choice/> or <sequence/>) container.
AddAttribute(CS.xsd, toWhat, parentComponentPath, precedingComponentPath, id, name, type, default, fixed, use, form, ref)	Adds the <attribute/> element with specified id, name, type, default, fixed, use, form, and ref to the toWhat (i.e. <attributeGroup/>, <complexType/>, <extension/>, or <schema/>) container, at the position defined by the path of its parent XML Schema component (parentComponentPath) and the path of its preceding XML Schema component (precedingComponentPath). If this latter is not specified by the designer, it means that the <attribute/> element is the first element in the toWhat container (or the parent XML Schema component).
DeleteAttribute(CS.xsd, fromWhat, attributePath)	Removes the <attribute/> element located at the position attributePath, from the fromWhat (i.e. <attributeGroup/>, <complexType/>, <extension/>, or <schema/>) container.
SetAttribute(CS.xsd, inWhat, attributePath, id, name, type, default, fixed, use, form, ref)	Changes (or introduces) id, name, type, default, fixed, use, form, or ref attributes of the <attribute/> element located at the position attributePath, in the inWhat (i.e. <attributeGroup/>, <complexType/>, <extension/>, or <schema/>) container.
AddAttributeGroup(CS.xsd, parentComponentPath, precedingComponentPath, id, name, ref)	Adds the <attributeGroup/> element with specified id, name, and ref to the toWhat (i.e. <attributeGroup/>, <complexType/>, <extension/>, <redefine/>, or <schema/>) container, at the position defined by the path of its parent XML Schema component (parentComponentPath) and the path of its preceding XML Schema component (precedingComponentPath). If this latter is not specified by the designer, it means that the <attributeGroup/> element is the first element in the toWhat container (or the parent XML Schema component).
DeleteAttributeGroup(CS.xsd, fromWhat, attributeGroupPath)	Removes the <attributeGroup/> element located at the position attributeGroupPath, from the fromWhat (i.e. <attributeGroup/>, <complexType/>, <extension/>, <redefine/>, or <schema/>) container.
SetAttributeGroup(CS.xsd, inWhat, attributeGroupPath, id, name, ref)	Changes (or introduces) id, name, or ref attributes of the <attributeGroup/> element located at the position attributeGroupPath, in the inWhat (i.e. <attributeGroup/>, <complexType/>, <extension/>, <redefine/>, or <schema/>) container.
AddAnyAttribute(CS.xsd, toWhat, parentComponentPath, precedingComponentPath, id, namespace, processContents)	Adds the <anyAttribute/> element with specified id, namespace and processContents to the toWhat (i.e. <attributeGroup/>, <complexType/>, or <extension/>) container, at the position defined by the path of its parent XML Schema component (parentComponentPath) and the path of its preceding XML Schema component (precedingComponentPath). If this latter is not specified by the designer, it means that the <anyAttribute/> element is the first element in the toWhat container (or the parent XML Schema component).
DeleteAnyAttribute(CS.xsd, fromWhat, anyAttributePath)	Removes the <anyAttribute/> element located at the position anyAttributePath, from the fromWhat (i.e. <attributeGroup/>, <complexType/>, or <extension/>) container.
SetAnyAttribute(CS.xsd, inWhat, anyAttributePath, id, namespace, processContents)	Changes (or introduces) id, namespace, or processContents attributes of the <anyAttribute/> element located at the position anyAttributePath, in the inWhat (i.e. <attributeGroup/>, <complexType/>, or <extension/>) container.
AddComplexType(CS.xsd, toWhat, parentComponentPath, precedingComponentPath, id, name, default, abstract, final, mixed, block)	Adds the <complexType/> element with specified id, name, default, abstract, final, mixed, and block to the toWhat (i.e. <element/>, <redefine/>, or <schema/>) container, at the position defined by the path of its parent XML Schema component (parentComponentPath) and the path of its preceding XML Schema component (precedingComponentPath). If this latter is not specified by the designer, it means that the <complexType/> element is the first element in the toWhat container (or the parent XML Schema component).

DeleteComplexType(CS.xsd, fromWhat, complexTypePath)	Removes the <complexType/> element located at the position complexTypePath, from the fromWhat (i.e. <element/>, <redefine/>, or <schema/>) container.
SetComplexType(CS.xsd, inWhat, complexTypePath, id, name, default, abstract, final, mixed, block)	Changes (or introduces) id, name, default, abstract, final, mixed, or block attributes of the <complexType/> element located at the position complexTypePath, in the inWhat (i.e. <element/>, <redefine/>, or <schema/>) container.
AddAll(CS.xsd, toWhat, parentComponentPath, precedingComponentPath, id, minOccurs, maxOccurs)	Adds the <all/> element with specified id, minOccurs and maxOccurs to the toWhat (i.e. <complexType/> or <group/>) container, at the position defined by the path of its parent XML Schema component (parentComponentPath) and the path of its preceding XML Schema component (precedingComponentPath). If this latter is not specified by the designer, it means that the <all/> element is the first element in the toWhat container (or the parent XML Schema component).
DeleteAll(CS.xsd, fromWhat, allPath)	Removes the <all/> element located at the position allPath, from the fromWhat (i.e. <complexType/> or <group/>) container.
SetAll(CS.xsd, inWhat, allPath, id, minOccurs, maxOccurs)	Changes (or introduces) id, minOccurs or maxOccurs attributes of the <all/> element located at the position allPath, in the inWhat (i.e. <complexType/> or <group/>) container.
AddChoice(CS.xsd, toWhat, parentComponentPath, precedingComponentPath, id, minOccurs, maxOccurs)	Adds the <choice/> element with specified id, minOccurs and maxOccurs to the toWhat (i.e. <complexType/>, <choice/>, <sequence/> or <group/>) container, at the position defined by the path of its parent XML Schema component (parentComponentPath) and the path of its preceding XML Schema component (precedingComponentPath). If this latter is not specified by the designer, it means that the <choice/> element is the first element in the toWhat container (or the parent XML Schema component).
DeleteChoice(CS.xsd, fromWhat, choicePath)	Removes the <choice/> element located at the position choicePath, from the fromWhat (i.e. <complexType/>, <choice/>, <sequence/> or <group/>) container.
SetChoice(CS.xsd, inWhat, choicePath, id, minOccurs, maxOccurs)	Changes (or introduces) id, minOccurs, or maxOccurs attributes of the <choice/> element located at the position choicePath, in the inWhat (i.e. <complexType/>, <choice/>, <sequence/> or <group/>) container.
AddSequence(CS.xsd, toWhat, parentComponentPath, precedingComponentPath, id, minOccurs, maxOccurs)	Adds the <sequence/> element with specified id, minOccurs, and maxOccurs to the toWhat (i.e. <complexType/>, <choice/>, <sequence/> or <group/>) container, at the position defined by the path of its parent XML Schema component (parentComponentPath) and the path of its preceding XML Schema component (precedingComponentPath). If this latter is not specified by the designer, it means that the <sequence/> element is the first element in the toWhat container (or the parent XML Schema component).
DeleteSequence(CS.xsd, fromWhat, sequencePath)	Removes the <sequence/> element located at the position sequencePath, from the fromWhat (i.e. <complexType/>, <choice/>, <sequence/> or <group/>) container.
SetSequence(CS.xsd, inWhat, sequencePath, id, minOccurs, maxOccurs)	Changes (or introduces) id, minOccurs, or maxOccurs attributes of the <sequence/> element located at the position sequencePath, in the inWhat (i.e. <complexType/>, <choice/>, <sequence/> or <group/>) container.
AddComplexContent(CS.xsd, complexTypePath, precedingComponentPath, id, mixed)	Adds the <complexContent/> element with specified id and mixed to the <complexType/> container, at the position defined by the path of its parent XML Schema component (complexTypePath) and the path of its preceding XML Schema component (precedingComponentPath). If this latter is not specified by the designer, it means that the <complexContent/> element is the first element in the <complexType/> container (or the parent XML Schema component).
DeleteComplexContent(CS.xsd, complexContentPath)	Removes the <complexContent/> element located at the position complexContentPath, from the <complexType/> container.

SetComplexContent(CS.xsd, complexContentPath, id, mixed)	Changes (or introduces) id or mixed attributes of the <complexContent/> element located at the position complexContentPath, in the <complexType/> container.
AddSimpleContent(CS.xsd, complexTypePath, precedingComponentPath, id)	Adds the <simpleContent/> element with specified id to the <complexType/> container, at the position defined by the path of its parent XML Schema component (complexTypePath) and the path of its preceding XML Schema component (precedingComponentPath). If this latter is not specified by the designer, it means that the <simpleContent/> element is the first element in the <complexType/> container (or the parent XML Schema component).
DeleteSimpleContent(CS.xsd, simpleContentPath)	Removes the <simpleContent/> element located at the position simpleContentPath, from the <complexType/> container.
SetSimpleContent(CS.xsd, simpleContentPath, id)	Changes (or introduces) id attribute of the <simpleContent/> element located at the position simpleContentPath, in the <complexType/> container.
AddExtension(CS.xsd, toWhat, parentComponentPath, precedingComponentPath, id, base)	Adds the <extension/> element with specified id and base to the toWhat (i.e. <simpleContent/> or <complexContent/>) container, at the position defined by the path of its parent XML Schema component (parentComponentPath) and the path of its preceding XML Schema component (precedingComponentPath). If this latter is not specified by the designer, it means that the <extension/> element is the first element in the toWhat container (or the parent XML Schema component).
DeleteExtension(CS.xsd, fromWhat, extensionPath)	Removes the <extension/> element located at the position extensionPath, from the fromWhat (i.e. <simpleContent/> or <complexContent/>) container.
SetExtension(CS.xsd, inWhat, extensionPath, id, base)	Changes (or introduces) id or base attributes of the <extension/> element located at the position extensionPath, in the inWhat (i.e. <simpleContent/> or <complexContent/>) container.
AddSimpleType(CS.xsd, toWhat, parentComponentPath, precedingComponentPath, id, name, final)	Adds the <simpleType/> element with specified id, name, and final to the toWhat (i.e. <attribute/>, <element/>, <redefine/>, <list/>, <union/>, <restriction/>, or <schema/>) container, at the position defined by the path of its parent XML Schema component (parentComponentPath) and the path of its preceding XML Schema component (precedingComponentPath). If this latter is not specified by the designer, it means that the <simpleType/> element is the first element in the toWhat container (or the parent XML Schema component).
DeleteSimpleType(CS.xsd, fromWhat, simpleTypePath)	Removes the <simpleType/> element located at the position simpleTypePath, from the fromWhat (i.e. <attribute/>, <element/>, <redefine/>, <list/>, <union/>, <restriction/>, or <schema/>) container.
SetSimpleType(CS.xsd, inWhat, simpleTypePath, id, name, final)	Changes (or introduces) id, name, or final attributes of the <simpleType/> element located at the position simpleTypePath, in the inWhat (i.e. <attribute/>, <element/>, <redefine/>, <list/>, <union/>, <restriction/>, or <schema/>) container.
AddList(CS.xsd, simpleTypePath, precedingComponentPath, id, itemType)	Adds the <list/> element with specified id and itemType to the <simpleType/> container, at the position defined by the path of its parent XML Schema component (simpleTypePath) and the path of its preceding XML Schema component (precedingComponentPath). If this latter is not specified by the designer, it means that the <list/> element is the first element in the <simpleType/> container (or the parent XML Schema component).

DeleteList(CS.xsd, listPath)	Removes the <list/> element located at the position listPath, from the <simpleType/> container.
SetList(CS.xsd, listPath, id, itemType)	Changes (or introduces) id or itemType attributes of the <list/> element located at the position listPath, in the <simpleType/> container.
AddUnion(CS.xsd, simpleTypePath, precedingComponentPath, id, memberTypes)	Adds the <union/> element with specified id and memberTypes to the <simpleType/> container, at the position defined by the path of its parent XML Schema component (simpleTypePath) and the path of its preceding XML Schema component (precedingComponentPath). If this latter is not specified by the designer, it means that the <union/> element is the first element in the <simpleType/> container (or the parent XML Schema component).
DeleteUnion(CS.xsd, unionPath)	Removes the <union/> element located at the position unionPath, from the <simpleType/> container.
SetUnion(CS.xsd, unionPath, id, memberTypes)	Changes (or introduces) id or memberTypes attributes of the <union/> element located at the position unionPath, in the <simpleType/> container.
AddRestriction(CS.xsd, toWhat, parentComponentPath, precedingComponentPath, id, base)	Adds the <restriction/> element with specified id and base to the toWhat (i.e. <simpleType/>, <simpleContent/>, or <complexContent/>) container, at the position defined by the path of its parent XML Schema component (parentComponentPath) and the path of its preceding XML Schema component (precedingComponentPath). If this latter is not specified by the designer, it means that the <restriction/> element is the first element in the toWhat container (or the parent XML Schema component).
DeleteRestriction(CS.xsd, fromWhat, restrictionPath)	Removes the <restriction/> element located at the position restrictionPath, from the fromWhat (i.e. <simpleType/>, <simpleContent/>, or <complexContent/>) container.
SetRestriction(CS.xsd, inWhat, restrictionPath, id, base)	Changes (or introduces) id or base attributes of the <restriction/> element located at the position restrictionPath, in the inWhat (i.e. <simpleType/>, <simpleContent/>, or <complexContent/>) container.
AddEnumeration(CS.xsd, restrictionPath, precedingComponentPath, id, value)	Adds the <enumeration/> element with specified id and value to the <restriction/> container, at the position defined by the path of its parent XML Schema component (restrictionPath) and the path of its preceding XML Schema component (precedingComponentPath). If this latter is not specified by the designer, it means that the <enumeration/> element is the first element in the <restriction/> container (or the parent XML Schema component).
DeleteEnumeration(CS.xsd, enumerationPath)	Removes the <enumeration/> element located at the position enumerationPath, from the <restriction/> container.
SetEnumeration(CS.xsd, enumerationPath, id, value)	Changes (or introduces) id or value attributes of the <enumeration/> element located at the position enumerationPath, in the <restriction/> container.
AddPattern(CS.xsd, restrictionPath, precedingComponentPath, id, value)	Adds the <pattern/> element with specified id and value to the <restriction/> container, at the position defined by the path of its parent XML Schema component (restrictionPath) and the path of its preceding XML Schema component (precedingComponentPath). If this latter is not specified by the designer, it means that the <pattern/> element is the first element in the <restriction/> container (or the parent XML Schema component).
DeletePattern(CS.xsd, patternPath)	Removes the <pattern/> element located at the position patternPath, from the <restriction/> container.
SetPattern(CS.xsd, patternPath, id, value)	Changes (or introduces) id or value attributes of the <pattern/> element located at the position patternPath, in the <restriction/> container.
AddTotalDigits(CS.xsd, restrictionPath, precedingComponentPath, id, value, fixed)	Adds the <totalDigits/> element with specified id, value, and fixed to the <restriction/> container, at the position defined by the path of its parent XML Schema component (restrictionPath) and the path of its preceding XML Schema component (precedingComponentPath). If this latter is not specified by the designer, it means that the <totalDigits/> element is the first element in the <restriction/> container (or the parent XML Schema component).

DeleteTotalDigits(CS.xsd, totalDigitsPath)	Removes the <totalDigits/> element located at the position totalDigitsPath, from the <restriction/> container.
SetTotalDigits(CS.xsd, totalDigitsPath, id, value, fixed)	Changes (or introduces) id, value, or fixed attributes of the <totalDigits/> element located at the position totalDigitsPath, in the <restriction/> container.
AddFractionDigits(CS.xsd, restrictionPath, precedingComponentPath, id, value, fixed)	Adds the <fractionDigits/> element with specified id, value, and fixed to the <restriction/> container, at the position defined by the path of its parent XML Schema component (restrictionPath) and the path of its preceding XML Schema component (precedingComponentPath). If this latter is not specified by the designer, it means that the <fractionDigits/> element is the first element in the <restriction/> container (or the parent XML Schema component).
DeleteFractionDigits(CS.xsd, fractionDigitsPath)	Removes the <fractionDigits/> element located at the position fractionDigitsPath, from the <restriction/> container.
SetFractionDigits(CS.xsd, fractionDigitsPath, id, value, fixed)	Changes (or introduces) id, value, or fixed attributes of the <fractionDigits/> element located at the position fractionDigitsPath, in the <restriction/> container.
AddMinInclusive(CS.xsd, restrictionPath, precedingComponentPath, id, value, fixed)	Adds the <minInclusive/> element with specified id, value, and fixed to the <restriction/> container, at the position defined by the path of its parent XML Schema component (restrictionPath) and the path of its preceding XML Schema component (precedingComponentPath). If this latter is not specified by the designer, it means that the <minInclusive/> element is the first element in the <restriction/> container (or the parent XML Schema component).
DeleteMinInclusive(CS.xsd, minInclusivePath)	Removes the <minInclusive/> element located at the position minInclusivePath, from the <restriction/> container.
SetMinInclusive(CS.xsd, minInclusivePath, id, value, fixed)	Changes (or introduces) id, value, or fixed attributes of the <minInclusive/> element located at the position minInclusivePath, in the <restriction/> container.
AddMaxInclusive(CS.xsd, restrictionPath, precedingComponentPath, id, value, fixed)	Adds the <maxInclusive/> element with specified id, value, and fixed to the <restriction/> container, at the position defined by the path of its parent XML Schema component (restrictionPath) and the path of its preceding XML Schema component (precedingComponentPath). If this latter is not specified by the designer, it means that the <maxInclusive/> element is the first element in the <restriction/> container (or the parent XML Schema component).
DeleteMaxInclusive(CS.xsd, maxInclusivePath)	Removes the <maxInclusive/> element located at the position maxInclusivePath, from the <restriction/> container.
SetMaxInclusive(CS.xsd, maxInclusivePath, id, value, fixed)	Changes (or introduces) id, value, or fixed attributes of the <maxInclusive/> element located at the position maxInclusivePath, in the <restriction/> container.
AddMinExclusive(CS.xsd, restrictionPath, precedingComponentPath, id, value, fixed)	Adds the <minExclusive/> element with specified id, value, and fixed to the <restriction/> container, at the position defined by the path of its parent XML Schema component (restrictionPath) and the path of its preceding XML Schema component (precedingComponentPath). If this latter is not specified by the designer, it means that the <minExclusive/> element is the first element in the <restriction/> container (or the parent XML Schema component).
DeleteMinExclusive(CS.xsd, minExclusivePath)	Removes the <minExclusive/> element located at the position minExclusivePath, from the <restriction/> container

SetMinExclusive(CS.xsd, minExclusivePath, id, value, fixed)	Changes (or introduces) id, value, or fixed attributes of the <minExclusive/> element located at the position minExclusivePath, in the <restriction/> container.
AddMaxExclusive(CS.xsd, restrictionPath, precedingComponentPath, id, value, fixed)	Adds the <maxExclusive/> element with specified id, value, and fixed to the <restriction/> container, at the position defined by the path of its parent XML Schema component (restrictionPath) and the path of its preceding XML Schema component (precedingComponentPath). If this latter is not specified by the designer, it means that the <maxExclusive/> element is the first element in the <restriction/> container (or the parent XML Schema component).
DeleteMaxExclusive(CS.xsd, maxExclusivePath)	Removes the <maxExclusive/> element located at the position maxExclusivePath, from the <restriction/> container.
SetMaxExclusive(CS.xsd, maxExclusivePath, id, value, fixed)	Changes (or introduces) id, value, or fixed attributes of the <maxExclusive/> element located at the position maxExclusivePath, in the <restriction/> container.
AddMinLength(CS.xsd, restrictionPath, precedingComponentPath, id, value, fixed)	Adds the <minLength/> element with specified id, value, and fixed to the <restriction/> container, at the position defined by the path of its parent XML Schema component (restrictionPath) and the path of its preceding XML Schema component (precedingComponentPath). If this latter is not specified by the designer, it means that the <minLength/> element is the first element in the <restriction/> container (or the parent XML Schema component).
DeleteMinLength(CS.xsd, minLengthPath)	Removes the <minLength/> element located at the position minLengthPath, from the <restriction/> container.
SetMinLength(CS.xsd, minLengthPath, id, value, fixed)	Changes (or introduces) id, value, or fixed attributes of the <minLength/> element located at the position minLengthPath, in the <restriction/> container.
AddMaxLength(CS.xsd, restrictionPath, precedingComponentPath, id, value, fixed)	Adds the <maxLength/> element with specified id, value, and fixed to the <restriction/> container, at the position defined by the path of its parent XML Schema component (restrictionPath) and the path of its preceding XML Schema component (precedingComponentPath). If this latter is not specified by the designer, it means that the <maxLength/> element is the first element in the <restriction/> container (or the parent XML Schema component).
DeleteMaxLength(CS.xsd, maxLengthPath)	Removes the <maxLength/> element located at the position maxLengthPath, from the <restriction/> container.
SetMaxLength(CS.xsd, maxLengthPath, id, value, fixed)	Changes (or introduces) id, value, or fixed attributes of the <maxLength/> element located at the position maxLengthPath, in the <restriction/> container.
AddLength(CS.xsd, restrictionPath, precedingComponentPath, id, value, fixed)	Adds the <length/> element with specified id, value, and fixed to the <restriction/> container, at the position defined by the path of its parent XML Schema component (restrictionPath) and the path of its preceding XML Schema component (precedingComponentPath). If this latter is not specified by the designer, it means that the <length/> element is the first element in the <restriction/> container (or the parent XML Schema component).
DeleteLength(CS.xsd, lengthPath)	Removes the <length/> element located at the position lengthPath, from the <restriction/> container.
SetLength(CS.xsd, lengthPath, id, value, fixed)	Changes (or introduces) id, value, or fixed attributes of the <length/> element located at the position lengthPath, in the <restriction/> container.
AddWhiteSpace(CS.xsd, restrictionPath, precedingComponentPath, id, value, fixed)	Adds the <whiteSpace/> element with specified id, value, and fixed to the <restriction/> container, at the position defined by the path of its parent XML Schema component (restrictionPath) and the path of its preceding XML Schema component (precedingComponentPath). If this latter is not specified by the designer, it means that the <whiteSpace/> element is the first element in the <restriction/> container (or the parent XML Schema component).

DeleteWhiteSpace(CS.xsd, whiteSpacePath)	Removes the <whiteSpace/> element located at the position whiteSpacePath, from the <restriction/> container.
SetWhiteSpace(CS.xsd, whiteSpacePath, id, value, fixed)	Changes (or introduces) id, value, or fixed attributes of the <whiteSpace/> element located at the position whiteSpacePath, in the <restriction/> container.
AddKey(CS.xsd, elementPath, precedingComponentPath, id, name)	Adds the <key/> element with specified id and name to the <element/> container, at the position defined by the path of its parent XML Schema component (elementPath) and the path of its preceding XML Schema component (precedingComponentPath). If this latter is not specified by the designer, it means that the <key/> element is the first element in the <element/> container (or the parent XML Schema component).
DeleteKey(CS.xsd, keyPath)	Removes the <key/> element located at the position keyPath, from the <element/> container.
SetKey(CS.xsd, keyPath, id, name)	Changes (or introduces) id or name attributes of the <key/> element located at the position keyPath, in the <element/> container.
AddKeyref(CS.xsd, elementPath, precedingComponentPath, id, name, refer)	Adds the <keyref/> element with specified id, name, and refer to the <element/> container, at the position defined by the path of its parent XML Schema component (elementPath) and the path of its preceding XML Schema component (precedingComponentPath). If this latter is not specified by the designer, it means that the <keyref/> element is the first element in the <element/> container (or the parent XML Schema component).
DeleteKeyref(CS.xsd, keyrefPath)	Removes the <keyref/> element located at the position keyrefPath, from the <element/> container.
SetKeyref(CS.xsd, keyrefPath, id, name, refer)	Changes (or introduces) id, name, or refer attributes of the <keyref/> element located at the position keyrefPath, in the <element/> container.
AddUnique(CS.xsd, elementPath, precedingComponentPath, id, name)	Adds the <unique/> element with specified id and name to the <element/> container, at the position defined by the path of its parent XML Schema component (elementPath) and the path of its preceding XML Schema component (precedingComponentPath). If this latter is not specified by the designer, it means that the <unique/> element is the first element in the <element/> container (or the parent XML Schema component).
DeleteUnique(CS.xsd, uniquePath)	Removes the <unique/> element located at the position uniquePath, from the <element/> container.
SetUnique(CS.xsd, uniquePath, id, name)	Changes (or introduces) id or name attributes of the <unique/> element located at the position uniquePath, in the <element/> container.
AddSelector(CS.xsd, toWhat, parentComponentPath, precedingComponentPath, id, xpath)	Adds the <selector/> element with specified id and xpath to the toWhat (i.e. <key/>, <keyref/>, or <unique/>) container, at the position defined by the path of its parent XML Schema component (parentComponentPath) and the path of its preceding XML Schema component (precedingComponentPath). If this latter is not specified by the designer, it means that the <selector/> element is the first element in the toWhat container (or the parent XML Schema component).
DeleteSelector(CS.xsd, fromWhat, selectorPath)	Removes the <selector/> element located at the position selectorPath, from the fromWhat (i.e. <key/>, <keyref/>, or <unique/>) container.
SetSelector(CS.xsd, inWhat, selectorPath, id, xpath)	Changes (or introduces) id or xpath attributes of the <selector/> element located at the position selectorPath, in the inWhat (i.e. <key/>, <keyref/>, or <unique/>) container.
AddField(CS.xsd, toWhat, parentComponentPath, precedingComponentPath, id, xpath)	Adds the <field/> element with specified id and xpath to the toWhat (i.e. <key/>, <keyref/>, or <unique/>) container, at the position defined by the path of its parent XML Schema component (parentComponentPath) and the path of its preceding XML Schema component (precedingComponentPath). If this latter is not specified by the designer, it means that the <field/> element is the first element in the toWhat container (or the parent XML Schema component).

DeleteField(CS.xsd, fromWhat, fieldPath)	Removes the <field/> element located at the position fieldPath, from the fromWhat (i.e. <key/>, <keyref/>, or <unique/>) container.
SetField(CS.xsd, inWhat, fieldPath, id, xpath)	Changes (or introduces) id or xpath attributes of the <field/> element located at the position fieldPath, in the inWhat (i.e. <key/>, <keyref/>, or <unique/>) container.
AddNotation(CS.xsd, precedingComponentPath, id, name, public, system)	Adds the <notation/> element with specified id, name, public, and system to the <schema/> container, after the XML Schema component located at the position precedingComponentPath. If this position is not specified by the designer, it means that the <notation/> element is the first element in the <schema/> container.
DeleteNotation(CS.xsd, notationPath)	Removes the <notation/> element located at the position notationPath, from the <schema/> container.
SetNotation(CS.xsd, notationPath, id, name, public, system)	Changes (or introduces) id, name, public, or system attributes of the <notation/> element located at the position notationPath, in the <schema/> container.
AddAnnotation(CS.xsd, toWhat, parentComponentPath, id)	Adds the <annotation/> element with specified id to the toWhat (i.e. <all/>, <any/>, <anyAttribute/>, <attribute/>, <attributeGroup/>, <choice/>, <complexContent/>, <complexType/>, <element/>, <enumeration/>, <extension/>, <field/>, <fractionDigits/>, <group/>, <import/>, <include/>, <key/>, <keyref/>, <length/>, <list/>, <maxExclusive/>, <maxInclusive/>, <maxLength/>, <minExclusive/>, <minInclusive/>, <minLength/>, <pattern/>, <redefine/>, <restriction/>, <selector/>, <sequence/>, <simpleContent/>, <simpleType/>, <totalDigits/>, <union/>, <unique/>, <whiteSpace/>, <notation/>, or <schema/>) container, as the first element (since the annotation element appears always at the beginning of most XML Schema components) in its parent XML Schema component (or the toWhat container) located at the position parentComponentPath.
DeleteAnnotation(CS.xsd, fromWhat, annotationPath)	Removes the <annotation/> element located at the position annotationPath, from the fromWhat (i.e. <all/>, <any/>, <anyAttribute/>, <attribute/>, <attributeGroup/>, <choice/>, <complexContent/>, <complexType/>, <element/>, <enumeration/>, <extension/>, <field/>, <fractionDigits/>, <group/>, <import/>, <include/>, <key/>, <keyref/>, <length/>, <list/>, <maxExclusive/>, <maxInclusive/>, <maxLength/>, <minExclusive/>, <minInclusive/>, <minLength/>, <pattern/>, <redefine/>, <restriction/>, <selector/>, <sequence/>, <simpleContent/>, <simpleType/>, <totalDigits/>, <union/>, <unique/>, <whiteSpace/>, <notation/>, or <schema/>) container.
SetAnnotation(CS.xsd, inWhat, annotationPath, id)	Changes (or introduces) id attribute of the <annotation/> element located at the position annotationPath, in the inWhat (i.e. <all/>, <any/>, <anyAttribute/>, <attribute/>, <attributeGroup/>, <choice/>, <complexContent/>, <complexType/>, <element/>, <enumeration/>, <extension/>, <field/>, <fractionDigits/>, <group/>, <import/>, <include/>, <key/>, <keyref/>, <length/>, <list/>, <maxExclusive/>, <maxInclusive/>, <maxLength/>, <minExclusive/>, <minInclusive/>, <minLength/>, <pattern/>, <redefine/>, <restriction/>, <selector/>, <sequence/>, <simpleContent/>, <simpleType/>, <totalDigits/>, <union/>, <unique/>, <whiteSpace/>, <notation/>, or <schema/>) container.
AddAppinfo(CS.xsd, annotationPath, precedingComponentPath, source, content)	Adds the <appinfo/> element with specified source and content to the <annotation/> container, at the position defined by the path of its parent XML Schema component (annotationPath) and the path of its preceding XML Schema component (precedingComponentPath). If this latter is not specified by the designer, it means that the <appinfo/> element is the first element in the <annotation/> container (or the parent XML Schema component).
DeleteAppinfo(CS.xsd, appinfoPath)	Removes the <appinfo/> element located at the position appinfoPath, from the <annotation/> container.
SetAppinfo(CS.xsd, appinfoPath, source, content)	Changes (or introduces) source attribute or content of the <appinfo/> element located at the position appinfoPath, in the <annotation/> container.

AddDocumentation(CS.xsd, annotationPath, precedingComponentPath, source, xml:lang, content)	Adds the <documentation/> element with specified source, "xml:lang" and content to the <annotation/> container, at the position defined by the path of its parent XML Schema component (annotationPath) and the path of its preceding XML Schema component (precedingComponentPath). If this latter is not specified by the designer, it means that the <documentation/> element is the first element in the <annotation/> container (or the parent XML Schema component).
DeleteDocumentation(CS.xsd, documentationPath)	Removes the <documentation/> element located at the position documentationPath, from the <annotation/> container.
SetDocumentation(CS.xsd, documentationPath, source, xml:lang, content)	Changes (or introduces) source or "xml:lang" attributes or content of the <documentation/> element located at the position documentationPath, in the <annotation/> container.

The effect of the **CreateConventionalSchema(CS.xsd)** primitive, that is the contents of the CS.xsd file after its application, is as follows:

```
<xsd:schema        xmlns:xsd="http://www.w3.org/2001/XMLSchema">
</xsd:schema>
```

The effect of the **AddInclude(CS.xsd, precedingComponentPath, id, schemaLocation)** primitive, that is the contents of the CS.xsd file after its application, is as follows:

```
<xsd:schema             xmlns:xsd="http://www.w3.org/2001/XMLSchema"
                        targetNamespace="targetNamespace">
  <xsd:include id="id" schemaLocation="schemaLocation"/>
  ...
</xsd:schema>
```

The effect of the **AddElement(CS.xsd, toWhat, parentComponentPath, precedingComponentPath, id, name, type, default, fixed, abstract, final, ref, minOccurs, maxOccurs, block, form, nillable, substitutionGroup)** primitive, that is the contents of the CS.xsd file after its application, is as follows:

If **toWhat** = schema

```
<xsd:schema          xmlns:xsd="http://www.w3.org/2001/XMLSchema"
                     targetNamespace="targetNamespace">
   ...
   <xsd:element id="id" name="name" type="type" default="minOccurs"
                final="maxOccurs" fixed="minOccurs" abstract="abstract"
                substitutionGroup="substitutionGroup" block="block" form="form"
                nillable="nillable">
   </xsd:element>
   ...
</xsd:schema>
```

The effect of the **AddAny(CS.xsd, toWhat, parentComponentPath, precedingComponentPath, id, namespace, minOccurs, maxOccurs, processContents)** primitive, that is the contents of the CS.xsd file after its application, is as follows:

If **toWhat** = sequence

```
<xsd:schema          xmlns:xsd="http://www.w3.org/2001/XMLSchema"
                     targetNamespace="targetNamespace">
    ...
    <xsd:complexType>
        <xsd:sequence>
              <xsd:any      id="id" namespace="namespace"
                            processContents="processContents"
minOccurs="minOccurs"
                            maxOccurs="maxOccurs"/>
        </xsd:sequence>
    </xsd:complexType>
    ...
</xsd:schema>
```

5.3. Illustrative example

Let us resume the example of the section 2.2. Suppose that on September 1, 2010, the designer realize that they also need information about all financial transactions done on each bank account. Then, he/she changes the first version of the conventional schema by adding a complex element `<FinancialTransactions>` to the Acount element. This new element contains a sequence of another complex element `<FinancialTransaction>` which is composed of three simple elements: `<FinancialTransactionDate>`, `<FinancialTransactionType>`, and `<FinancialTransactionAmount>`.
The second version of the conventional schema is shown in Figure 4. Thus, the temporal schema is also updated by adding a new slice related to this new version of the conventional schema, as shown in Figure 5. Changes are presented in purple.
The sequence of primitives that have been performed on the first version of the conventional schema (Bank_V1.xsd) to produce the second one (Bank_V2.xsd) is as follows:

(i) **AddElement**("Bank_V1.xsd", sequence, "/schema/element[@name='Account']/complexType/sequence", "/schema/element[@name='Account']/complexType/sequence/element[@name='Balance']", , "FinancialTransactions", , , , , , , , , , ,)

(ii) **AddComplexType**("Bank_V1.xsd", element, "/schema/element[@name='Account']/complexType/sequence/element[@name='FinancialTransactions']" , , , , , , , ,)

(iii) **AddSequence**("Bank_V1.xsd", complexType, "/schema/element[@name='Account']/complexType/sequence/element[@name='FinancialTransactions']/complexType", , , ,)

(iv) **AddElement**("Bank_V1.xsd", sequence, "/schema/element[@name='Account']/complexType/sequence/element[@name='FinancialTransactions']/complexType/sequence", , , "FinancialTransaction", , , , , , , , unbounded, , , ,)

(v) **AddComplexType**("Bank_V1.xsd", element, "/schema/element[@name='Account']/complexType/sequence/element[@name='FinancialTransactions']/complexType/sequence/element[@name=' FinancialTransaction']", , , , , , , ,)

(vi) **AddSequence(**"Bank_V1.xsd", complexType, "/schema/element[@name='Account']/complexType/sequence/element[@name='FinancialTransactions']/complexType/sequence/element[@name=' FinancialTransaction']/complexType", , , ,)

(vii) **AddElement**(“Bank_V1.xsd”, sequence,
“/schema/element[@name=’Account’]/complexType/sequence/element[@name=’FinancialTransactions’]/complexType/sequence/element[@name=’ FinancialTransaction’]/complexType/sequence”, , ,
“FinancialTransactionDate”, date, , , , , , , , , ,)

(viii) **AddElement**(“Bank_V1.xsd”, sequence,
“/schema/element[@name=’Account’]/complexType/sequence/element[@name=’FinancialTransactions’]/complexType/sequence/element[@name=’ FinancialTransaction’]/complexType/sequence”,
“/schema/element[@name=’Account’]/complexType/sequence/element[@name=’FinancialTransactions’]/complexType/sequence/element[@name=’
FinancialTransaction’]/complexType/sequence/element[@name=’FinancialTransactionDate’]”, ,
“FinancialTransactionType”, string, , , , , , , , , ,)

(ix) **AddElement**(“Bank_V1.xsd”, sequence,
“/schema/element[@name=’Account’]/complexType/sequence/element[@name=’FinancialTransactions’]/complexType/sequence/element[@name=’ FinancialTransaction’]/complexType/sequence”,
“/schema/element[@name=’Account’]/complexType/sequence/element[@name=’FinancialTransactions’]/complexType/sequence/element[@name=’
FinancialTransaction’]/complexType/sequence/element[@name=’FinancialTransactiontType’]”, ,
“FinancialTransactionAmount”, float, , , , , , , , , ,)

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

Figure 4. Second version of the conventional schema (Bank_V2.xsd), on September 1, 2010

```
<?xml version="1.0" encoding="UTF-8"?>
<temporalSchema xmlns="http://www.cs.arizona.edu/tau/tauXSchema/TSSchema">
   <conventionalSchema>
        <sliceSequence>
           <slice location="Bank_V1.xsd" begin="2010-02-01" />
           <slice location="Bank_V2.xsd" begin="2010-09-01" />
        </sliceSequence>
   </conventionalSchema>
   <annotationSet>
        <sliceSequence>
           <slice location="BankAnnotations_V1.xml" begin="2010-02-01" />
           <slice location="BankAnnotations_V2.xml" begin="2010-06-01" />
           <slice location="BankAnnotations_V3.xml" begin="2010-08-01" />
        </sliceSequence>
   </annotationSet>
</temporalSchema>
```

Figure 5. Temporal schema (BankTemporalSchema.xml) on September 1, 2010

On March 1, 2011, suppose that the designer decides to make some changes to the conventional schema. In the <Account> element, he/she renames the <OwnerName> element to be <AccountHolderName>. He/She adds a new element, <FinancialTransactionInstant>, to the element <FinancialTransaction>, in order to notify the instant of the financial transaction. He/She replaces the "Number" attribute of the <Account> element by a new attribute named "BBAN", which denotes the "basic/national bank account number". Furthermore, he/she adds a new attribute named "IBAN" to the <Account> element, which denotes the "international bank account number".

The third version of the conventional schema is shown in Figure 6 and the updated temporal schema document is shown in Figure 7. Changes are presented in purple.

The sequence of primitives that have been performed on the second version of the conventional schema (Bank_V2.xsd) to produce the third one (Bank_V3.xsd) is as follows:

(i) **SetElement**("Bank_V2.xsd", sequence,
"/schema/element[@name='Account']/complexType/sequence/element[@name='OwnerName']",
unchanged, "AccountHolderName", unchanged, unchanged, unchanged, unchanged, unchanged,
unchanged, unchanged, unchanged, unchanged, unchanged, unchanged, unchanged)

(ii) **AddElement**("Bank_V2.xsd", sequence,
"/schema/element[@name='Account']/complexType/sequence/element[@name='FinancialTransactions']/
complexType/sequence/element[@name=' FinancialTransaction']/complexType/sequence",
"/schema/element[@name='Account']/complexType/sequence/element[@name='FinancialTransactions']/
complexType/sequence/element[@name='
FinancialTransaction']/complexType/sequence/element[@name='FinancialTransactionDate']", ,
FinancialTransactionInstant, time, , , , , , , , , ,)

(iii) **SetAttribute**("Bank_V2.xsd", complexType,
"/schema/element[@name='Account']/complexType/attribute[@name='Number']", unchanged, "BBAN",
positiveInteger, unchanged, unchanged, unchanged, unchanged, unchanged)

(iv) **AddAttribute**("Bank_V2.xsd", complexType, "/schema/element[@name='Account']/complexType",
"/schema/element[@name='Account']/complexType/attribute[@name='BBAN']", , "IBAN", string, , ,
required, ,)

```
<?xml version="1.0" encoding="UTF-8"?>
<schema xmlns="http://www.w3.org/2001/XMLSchema">
 <element name="Bank">
  <complexType>
   <sequence>
    <element ref="Account"/>
   </sequence>
  </complexType>
 </element>
 <element name="Account">
  <complexType>
   <sequence>
    <element name="AccountHolderName" type="string"/>
    <element name="OpeningDate" type="date"/>
    <element name="Type" type="string"/>
    <element name="Balance" type="float"/>
    <element name="FinancialTransactions">
     <complexType>
      <sequence>
       <element name="FinancialTransaction" maxOccurs="unbounded">
        <complexType>
         <sequence>
          <element name="FinancialTransactionDate" type="date"/>
          <element name="FinancialTransactionInstant" type="time"/>
          <element name="FinancialTransactionType" type="string"/>
          <element name="FinancialTransactionAmount" type="float"/>
         </sequence>
        </complexType>
       </element>
      </sequence>
     </complexType>
    </element>
   </sequence>
   <attribute name="BBAN" type="positiveInteger" use=”required”/>
   <attribute name="IBAN" type="string" use=”required”/>
  </complexType>
 </element>
</schema>
```

Figure 6. Third version of the conventional schema (Bank_V3.xsd), on March 1, 2011

```
<?xml version="1.0" encoding="UTF-8"?>
<temporalSchema xmlns="http://www.cs.arizona.edu/tau/tauXSchema/TSSchema">
   <conventionalSchema>
        <sliceSequence>
           <slice location="Bank_V1.xsd" begin="2010-02-01" />
           <slice location="Bank_V2.xsd" begin="2010-09-01" />
           <slice location="Bank_V3.xsd" begin="2011-03-01" />
        </sliceSequence>
   </conventionalSchema>
   <annotationSet>
        <sliceSequence>
           <slice location="BankAnnotations_V1.xml" begin="2010-02-01" />
           <slice location="BankAnnotations_V2.xml" begin="2010-06-01" />
           <slice location="BankAnnotations_V3.xml" begin="2010-08-01" />
        </sliceSequence>
   </annotationSet>
</temporalSchema>
```

Figure 7. Temporal schema (BankTemporalSchema.xml) on March 1, 2011

On April 1, 2012, suppose that the designer decides to make some other changes to the conventional schema:

- He/She changes the types of the elements `<Balance>` and `<FinancialTransactionAmount>`, which become “double”, and the type of the attribute `<BBAN>`, which becomes “string”.
- He/She replaces the two elements `<FinancialTransactionDate>` and

<FinancialTransactionInstant> by a single element <FinancialTransactionTime>, which denotes both the date and the instant of the financial transaction.

- He/She defines, at the end of the schema, two new simple types, "accountType" (for account types) and "financTransactType" (for financial transaction types), by restricting the existing simple type "string". The "accountType" type limits the "string" type to the following set of distinct values: "*deposit*", "*checking*", "*current*", "*personal*", and "*transaction deposit*". The "financTransactType" type limits the "string" type to the following set of distinct values: "*deposit*", and "*withdrawal*". After that, he/she changes the types of the elements <Type> and <FinancialTransactionType>, which become "accountType" and "financTransactType" respectively.

The fourth version of the conventional schema is shown in Figure 8 and the updated temporal schema document is shown in Figure 9. Changes are presented in purple.

The sequence of primitives that have been performed on the third version of the conventional schema (Bank_V3.xsd) to produce the fourth one (Bank_V4.xsd) is as follows:

(i) **SetElement**("Bank_V3.xsd", sequence,
"/schema/element[@name='Account']/complexType/sequence/element[@name='Balance']", unchanged, unchanged, double, unchanged, unchanged, unchanged, unchanged, unchanged, unchanged, unchanged, unchanged, unchanged, unchanged, unchanged)

(ii) **SetElement**("Bank_V3.xsd", sequence,
"/schema/element[@name='Account']/complexType/sequence/element[@name='FinancialTransactions']/complexType/sequence/element[@name='FinancialTransaction']/complexType/sequence/element[@name='FinancialTransactionAmount']", unchanged, unchanged, double, unchanged, unchanged, unchanged, unchanged, unchanged, unchanged, unchanged, unchanged, unchanged, unchanged, unchanged)

(iii) **SetAttribute**("Bank_V3.xsd", complexType,
"/schema/element[@name='Account']/complexType/attribute[@name='BBAN']", unchanged, unchanged, string, unchanged, unchanged, unchanged, unchanged, unchanged)

(iv) **SetElement**("Bank_V3.xsd", sequence,
"/schema/element[@name='Account']/complexType/sequence/element[@name='FinancialTransactions']/complexType/sequence/element[@name='FinancialTransaction']/complexType/sequence/element[@name='FinancialTransactionDate']", unchanged, "FinancialTransactionTime", dateTime, unchanged, unchanged, unchanged, unchanged, unchanged, unchanged, unchanged, unchanged, unchanged, unchanged, unchanged)

(v) **DeleteElement**("Bank_V3.xsd", sequence,
"/schema/element[@name='Account']/complexType/sequence/element[@name='FinancialTransactions']/complexType/sequence/element[@name='FinancialTransaction']/complexType/sequence/element[@name='FinancialTransactionInstant']")

(vi) **AddSimpleType**("Bank_V3.xsd", schema, "/schema", "/schema/element[@name='Account']", , accountType,)

(vii) **AddRestriction**("Bank_V3.xsd", simpleType, "/schema/simpleType[@name='accountType']", , , string)

(viii) **AddEnumeration**("Bank_V3.xsd", "/schema/simpleType[@name='accountType']/restriction", , , "deposit")

(ix) **AddEnumeration**("Bank_V3.xsd", "/schema/simpleType[@name='accountType']/restriction", "/schema/simpleType[@name='accountType']/restriction/enumeration[value='deposit']", , "checking")

(x) **AddEnumeration**(“Bank_V3.xsd”, “/schema/simpleType[@name=’accountType’]/restriction”, “/schema/simpleType[@name=’accountType’]/restriction/enumeration[value=’checking’]”, , “current”)

(xi) **AddEnumeration**(“Bank_V3.xsd”, “/schema/simpleType[@name=’accountType’]/restriction”, “/schema/simpleType[@name=’accountType’]/restriction/enumeration[value=’current’]”, , “personal”)

(xii) **AddEnumeration**(“Bank_V3.xsd”, “/schema/simpleType[@name=’accountType’]/restriction”, “/schema/simpleType[@name=’accountType’]/restriction/enumeration[value=’personal’]”, , “transaction deposit”)

(xiii) **AddSimpleType**(“Bank_V3.xsd”, schema, “/schema”, “/schema/simpleType[@name=’accountType’]”, , financTransactType,)

(xiv) **AddRestriction**(“Bank_V3.xsd”, simpleType, “/schema/simpleType[@name=’ financTransactType’]”, , , string)

(xv) **AddEnumeration**(“Bank_V3.xsd”, “/schema/simpleType[@name=’ financTransactType’]/restriction”, , , “deposit”)

(xvi) **AddEnumeration**(“Bank_V3.xsd”, “/schema/simpleType[@name=’ financTransactType’]/restriction”, “/schema/simpleType[@name=’ financTransactType’]/restriction/enumeration[value=’deposit’]”, , “withdrawal”)

(xvii) **SetElement**(“Bank_V3.xsd”, sequence, “/schema/element[@name=’Account’]/complexType/sequence/element[@name=’Type’]”, unchanged, unchanged, accountType, unchanged, unchanged, unchanged, unchanged, unchanged, unchanged, unchanged, unchanged, unchanged, unchanged, unchanged)

(xviii) **SetElement**(“Bank_V3.xsd”, sequence, “/schema/element[@name=’Account’]/complexType/sequence/element[@name=’FinancialTransactions’]/complexType/sequence/element[@name=’FinancialTransaction’]/complexType/sequence/element[@name=’FinancialTransactionType’]”, unchanged, unchanged, financTransactType, unchanged, unchanged, unchanged, unchanged, unchanged, unchanged, unchanged, unchanged, unchanged, unchanged, unchanged)

```
<?xml version="1.0" encoding="UTF-8"?>
<schema xmlns="http://www.w3.org/2001/XMLSchema">
 <element name="Bank">
  <complexType>
   <sequence>
    <element ref="Account"/>
   </sequence>
  </complexType>
 </element>
 <element name="Account">
  <complexType>
   <sequence>
    <element name="AccountHolderName" type="string"/>
    <element name="OpeningDate" type="date"/>
    <element name="Type" type="accountType"/>
    <element name="Balance" type="double"/>
    <element name="FinancialTransactions">
     <complexType>
      <sequence>
       <element name="FinancialTransaction" maxOccurs="unbounded">
        <complexType>
         <sequence>
          <element name="FinancialTransactionTime" type="dateTime"/>
          <element name="FinancialTransactionType" type="financTransactType"/>
          <element name="FinancialTransactionAmount" type="double"/>
         </sequence>
        </complexType>
       </element>
      </sequence>
     </complexType>
    </element>
   </sequence>
```

```
    <attribute name="BBAN" type="string" use="required"/>
    <attribute name="IBAN" type="string" use="required"/>
   </complexType>
  </element>
  <simpleType name="accountType">
   <restriction base="string"/>
    <enumeration value="deposit"/>
    <enumeration value="checking"/>
    <enumeration value="current"/>
    <enumeration value="personal"/>
    <enumeration value="transaction deposit"/>
   </restriction>
  </simpleType>
  <simpleType name="financTransactType">
   <restriction base="string"/>
    <enumeration value="deposit"/>
    <enumeration value="withdrawal"/>
   </restriction>
  </simpleType>
</schema>
```

Figure 8. Fourth version of the conventional schema (Bank_V4.xsd), on April 1, 2012

```
<?xml version="1.0" encoding="UTF-8"?>
<temporalSchema xmlns="http://www.cs.arizona.edu/tau/tauXSchema/TSSchema">
   <conventionalSchema>
        <sliceSequence>
           <slice location="Bank_V1.xsd" begin="2010-02-01" />
           <slice location="Bank_V2.xsd" begin="2010-09-01" />
           <slice location="Bank_V3.xsd" begin="2011-03-01" />
           <slice location="Bank_V4.xsd" begin="2012-04-01" />
        </sliceSequence>
   </conventionalSchema>
   <annotationSet>
        <sliceSequence>
           <slice location="BankAnnotations_V1.xml" begin="2010-02-01" />
           <slice location="BankAnnotations_V2.xml" begin="2010-06-01" />
           <slice location="BankAnnotations_V3.xml" begin="2010-08-01" />
        </sliceSequence>
    </annotationSet>
</temporalSchema>
```

Figure 9. Temporal schema (BankTemporalSchema.xml) on April 1, 2012

6. Related Work

Schema versioning in the τXSchema [6,7,8,11] framework means versioning of conventional schema and versioning of annotations. In [9,12], the authors study versioning of annotations. They propose a complete set of change primitives for physical and logical annotations and define their operational semantics. The present work completes the picture by studying conventional schema versioning and by proposing a set of primitives for supporting also the evolution of this component of the τXSchema framework.

In [13], the authors propose six generic operations for XML schema change; three operations act on an XML schema element (i.e. addition, deletion and modification of an XML Schema element) and three operations act on an attribute in an XML schema element (i.e. addition, deletion and modification of an attribute). In this work, a completion of the work started in [13] is done in the context of the τXSchema approach, at a deeper and more detailed level (in the end, we introduced one hundred and twenty primitives). In fact, we investigate changes to all elements that belong to definition of the XML Schema language [10].

In [6], the authors introduce τXSchema but do not discuss schema versioning. In [7], [8] and [11], the authors deal with schema versioning in τXSchema, but focus only on capturing a time-varying schema

and validating documents against such a schema. All these previous works on τXSchema (i.e. [6], [7], [8], and [11]) do not study how the schema changes are performed, or what schema change operations are supported.

In [14], a set of primitives for updating XML schema has been defined. But these primitives deal only with simple types, complex types and elements. Our work is both more detailed and more global since it proposes primitives for changing all components of an XML Schema.

In [15], the authors present X-Evolution which is a web-based tool making the primitives defined in [14] available to the user both through a graphical interface and through a specifically tailored schema update language named XSchemaUpdate. In [16] and [17], the authors present EXup which is an engine for specifying XSchemaUpdate statements, translating them in XQuery Update Facility [18] expressions and evaluating them against XML Schema and associated documents.

In [19], the authors propose a set of rules and an algorithm for reducing sequences of XML Schema (or XML documents) updates: reducing a sequence of updates on an XML Schema (or an XML document) tree means deriving a shorter sequence with the same effect on this tree.

As surveyed in [20], updates on XML Schema have received less attention by database research community despite of their importance in XML databases. Furthermore, the authors show that the support of XML schema updates is absent in commercial XML tools (like Stylus Studio or XML Spy) and limited in commercial DBMSs (like Oracle 11g, Tamino, or DB2 v.9). Our work deals with the issue of XML Schema updates in an environment that supports XML Schema versioning.

7. Conclusion

In this work, we focused on the versioning of conventional schema in the τXSchema framework. In particular, we proposed a sound and complete set of primitives allowing the designer to define and to make changes on conventional schema; the syntax and operational semantics of each primitive have been defined.

We think that even if we have not considered the full XML Schema definition (e.g., involving the latest W3C Part 1: Structures and Part 2: Datatypes recommendations), we have addressed the XML Schema detail at a reasonable depth, since our work considers all the features listed in the XML Schema Part 0: Primer [10].

Currently, we are integrating our previous work on τXSchema [9,12] with the present one, in order to have a complete and consistent approach for schema versioning in τXSchema-based multitemporal XML repositories.

As for future work, we aim at extending our approach by studying temporal queries across schema versions in the τXSchema framework. To do this, we will start from the τXQuery language [21], which allows user to perform temporal queries in that framework, but without support for schema versioning.

8. References

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

[2] Zaniolo C., Wang F., "Temporal queries and version management in XML-based document archives", *Data and Knowledge Engineering*, 65(2), 2008, pp. 304-324.

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

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

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

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

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

[8] Dyreson C. E., Snodgrass R. T., Currim F., Currim S., Joshi S., "Validating Quicksand: Schema Versioning in τXSchema", *Data Knowledge and Engineering*, 65 (2), 2008, pp. 223-242.

[9] Brahmia Z., Bouaziz R., Grandi F., Oliboni B., "Schema Versioning in τXSchema-Based Multitemporal XML Repositories", *Proceedings of the 5th IEEE International Conference on Research Challenges in Information Science (RCIS 2011)*, Guadeloupe - French West Indies, France, 19-21 May 2011, pp. 1-12.

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

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

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

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

[14] Guerrini G., Mesiti M., Rossi D., "Impact of XML Schema Evolution on Valid Documents", *Proceedings of the 7th ACM International Workshop on Web Information and Data Management (WIDM 2005)*, Bermen, Germany, 5 November 2005, pp. 39-44.

[15] Guerrini G., Mesiti M., "X-Evolution: A Comprehensive Approach for XML Schema Evolution", *Proceedings of the 19th International Workshop Database and Expert Systems Applications (DEXA 2008)*, Turin, Italy, 1-5 September 2008, pp. 251-255.

[16] Cavalieri F., "EXup: an engine for the evolution of XML schemas and associated documents", *Proceedings of the 2010 EDBT/ICDT Workshops*, Lausanne, Switzerland, 22-26 March 2010.

[17] Cavalieri F., Guerrini G., Mesiti M., "Updating XML schemas and associated documents through exup", *Proceedings of the 27th International Conference on Data Engineering (ICDE 2011)*, Hannover, Germany, 11-16 April 2011, pp. 1320-1323.

[18] W3C, "XQuery Update Facility 1.0", *W3C Candidate Recommendation*, 17 March 2011. <http://www.w3.org/TR/2011/REC-xquery-update-10-20110317/>

[19] Cavalieri F., Guerrini G., Mesiti M., Oliboni B., "On the Minimization of Sequences of XML Document and Schema Update Operations", *Workshops Proceedings of the 27th International Conference on Data Engineering (ICDE 2011)*, Hannover, Germany, 11-16 April 2011, pp. 77-86.

[20] Colazzo D., Guerrini G., Mesiti M., Oliboni B., Waller E., "Document and Schema XML Updates", in Li C., Ling T. W., (Eds.), *Advanced Applications and Structures in XML Processing: Label Stream, Semantics Utilization and Data Query Technologies*, IGI Global, 2010, pp. 361-384.

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