A Systematic Approach for Changing XML Namespaces in XML Schemas and Managing their Effects on Associated XML Documents under Schema Versioning

Zouhaier Brahmia
Faculty of Economics and Management, University of Sfax
Tunisia
zouhaier.brahmia@fsegs.rnu.tn

Fabio Grandi
Dipartimento di Informatica – Scienza e Ingegneria, Alma Mater Studiorum – Università di Bologna
Italy

Rafik Bouaziz
Faculty of Economics and Management, University of Sfax
Tunisia

JDIM
Journal of Digital Information Management

ABSTRACT: *Although a lot of research work has been done concerning schema changes that deal with basic components of XML Schemas, like element and attribute declarations, simple types, and complex type definitions, the current literature of the XML database field does not include any paper that studies changes of advanced concepts of XML Schemas, like XML namespaces, local (or global) qualified (or unqualified) declarations, and schema definition styles (e.g., Russian Doll, Garden of Eden). In this paper, we study how XML namespace evolution is handled in an XML environment that supports both schema and document versioning. To the best of our knowledge, this is the first work on such a topic. Indeed, we propose a systematic approach that allows specifying changes on XML namespaces defined in XML Schemas, and propagating their effects to underlying XML documents, while maintaining all XML schema and XML instance versions.*

Categories and Subject Descriptors
H.2.1 [Database Management]: Logical Design—Schema and subschema; **H.2.7 [Database Management]:** Database Administration—Data dictionary/directory

General Terms:
Database Management, XML, XML Database

Keywords: XML database; XML Schema; XML namespace; Schema change; Schema change propagation; Schema versioning

Received: 16 August 2016, Revised 16 October 2016, Accepted 16 October 2016

DOI: 10.6025/jdim/2016/14/5/275-289

1. Introduction

In an XML database or repository [1, 2], both XML data and schemas tend to change over time [3, 4] for many reasons, including changes in the users' requirements, correction of mistakes in the initial design, and migration to a new platform. Many applications (like banking, flight reservation, e-business, e-government, and e-health) are temporal in nature [5, 6] and require a full history of data and schema changes for several reasons: avoiding loss of data after schema changes, maintenance of legacy data formatted according to past schemas, reuse of legacy applications, and auditing purposes. Such a history must be managed efficiently, consistently, and in a transparent way with regard to the end user. Besides, schema versioning [7, 8, 9] has long been advocated to be the more appropriate solution to support a complete data and schema history in databases.

In XML databases, several works have dealt with changes of basic concepts of XML Schemas [10] (e.g., element and attribute declarations, simple types, complex type definitions) [11-14]. However, there is no work that has dealt with changes involving advanced concepts of XML Schemas like XML namespaces, local/global qualified/unqualified declarations, and schema definition styles (e.g., Russian Doll, Salami, Bologna, Venetian Blind, Garden of Eden). In this paper, we deal with XML namespace evolution. An XML namespace [15-17] is a set of names, identified by a URI reference [18], which are used in XML documents as element types and attribute names; this set of names can evolve over time to respond to some users' requirements or to reflect some changes in the modeled reality.

To the best of our knowledge, we are the first to study such a topic. In fact, we propose a systematic approach for managing changes of XML namespaces, in an environment that supports XML schema versioning. Such an approach (i) provides a set of schema change operations that act on XML namespaces specified in XML Schemas, and (ii) defines the impact of each one of these operations on the underlying XML documents and, more precisely, on qualified/unqualified local/global definitions, while keeping track of all XML schema versions and their corresponding XML instance versions.

For the sake of simplicity, in our current work, we consider only simple XML documents with nested elements, attributes and textual contents without other kinds of features/constraints. In our future work, we will focus on more complicated aspects related to this topic.

In sum, as we introduced in the preliminary version of this work [19], we aim at proposing a generic approach that (i) helps XML database designers or administrators in the management of these complex schema changes, (ii) preserves the consistency of the database after the execution of any transaction which includes such schema change operations, and (iii) keeps track of all the produced schema and instance versions in a schema versioning framework. Furthermore, with respect to [19], in this paper we (i) extend the initial list of schema change operations with nine new operations, in order to provide more facilities to the database designer when dealing with XML namespace changes, (ii) expand the running example, in order to better illustrate our approach, by (a) giving more details on intermediate steps, (b) using some of the newly added operations, and (c) adding a new schema change involving namespaces, while showing its effects on the XML repository, and (iii) discuss related work while clarifying our scientific contribution with respect to both the state of the art and the state of the practice.

The rest of this paper is structured as follows. The next section first motivates the need for some automatic support to manage both changes of XML namespaces, in XML Schema documents, and their effects on XML Schema instances, and then introduces a motivating example. Section 3 presents the set of operations that we propose for changing XML namespaces and for consistently propagating their effects to the involved XML documents; an illustration of the use of these operations is shown through the reprise of the motivating example started in Section 2. Section 4 discusses related work. Section 5 summarizes the paper and gives some remarks about our future work.

2. Motivation

In this section, we first show the need for automatic support to help XML database designers in managing changes of XML namespaces in XML schemas and their effect on XML instances. Then we present an example that motivates our topic.

2.1. Need for Supporting Changes involving XML Namespaces and their Impact on XML Instances

By studying the state-of-the-art and the state-of-the-practice of XML schema evolution and versioning (e.g., [11-14], [20-24]), we notice that there is no work that has dealt with schema changes involving XML namespaces; all existing approaches and contributions handle schema changes involving basic concepts of XML schemas like XSD elements, attributes, sequences, simple types, and complex type definitions. Furthermore, we also notice that existing XML database management systems and XML-based tools do not provide any support for handling XML namespace changes.

Besides, the XML database designer or administrator would need automatic support for efficiently managing changes to XML namespaces, since such changes could happen in any application that exploits XML documents with their corresponding XML schema. For example, he/she could change a previous design option taken for XML instances of the first version (S_V1) of an XML Schema S, by specifying, in the new version of S (S_V2), that the appearance of locally declared elements and attributes in any XML document which is valid to this new version (i.e., S_V2) must be qualified by an XML namespace. Moreover, changes of XML namespaces, defined in XML schema documents, have several implications on the validation of existing XML instances documents, and require a lot of time and attention to consistently propagate changes performed at schema level to instance level.

Therefore, we think that it is very interesting and useful to study these aspects and to provide theoretical and practical solutions for the issue of computer-aided XML namespace evolution, especially when both XML schemas and XML instance versions have to be maintained. In fact, dealing with changes to XML namespaces, and resolving problems that result from these changes, aims at providing several engineering improvements including fostering interoperability and reuse of software, and facilitating cooperation between programmers with different habits and styles. The proposed solutions could be useful to practitioners meeting the pressing needs to provide

solutions and support tools for managing XML schema evolution/versioning in large web information systems (e.g., Wikipedia) [25], Big Science projects (e.g., Ensembl Genome) [26], and open-source highly collaborative software development environments (e.g., GitHub). Furthermore, considering XML schema designers, application developers, standards and open-source community members, who are working on cloud infrastructures and deal with manipulation of shared XML data and schemas, the following tasks [14] that they daily have to face are made easier:
(i) keeping track of all changes done on XML schemas,
(ii) maintaining (backward and forward) compatibility between schema versions and applications which exploit their instances, and
(iii) upgrading any given schema version.

2.2. Running Example

As a motivating and illustrative example for our approach, we recall and extend the example presented in the preliminary version of this work [19]. Indeed, suppose that, in order to improve scientific research in the country and to provide easy and quick access to all scientific productions, the Ministry of Higher Education and Research wants to bookkeep and publish information on all PhD theses, using XML documents. Suppose that the corresponding XML repository includes in its first state.

```
<?xml version="1.0" encoding="UTF-8"?>
 <xsd:schema
        xmlns:xsd="http://www.w3.org/2001/
XMLSchema"
        targetNamespace="http://www.thesis.org"
        xmlns:t="http://www.thesis.org"
        xmlns:foaf="http://xmlns.com/foaf/0.1/"
        xmlns:i="http://www.institution.com"
        attributeFormDefault="unqualified"
        elementFormDefault="unqualified">
              <xsd:importnamespace="http://
www.institution.com
        schemaLocation="Institution_V1.xsd"/>
<xsd:element name="thesis">
<xsd:complexType>
<xsd:sequence>
<xsd:element name="title" type="xsd:string"/>
<xsd:element name="student"
        type="foaf:Person"/>
<xsd:element name="advisor"
        type="foaf:Person"/>
<xsd:element name="university"
        type="i:institution"/>
</xsd:sequence>
<xsd:attribute name="ID" type="xsd:string"
use="required"/>
</xsd:complexType>
</xsd:element>
</xsd:schema>
```

Figure 1. The first schema version of theses (file Thesis_V1.xsd)

three resources: Thesis_V1.xsd (see Fig.1), Institution_V1.xsd (see Fig. 2), and Thesis_V1.xml (see Fig. 3).

```
<?xml version="1.0" encoding="UTF-8"?>
  <schema  xmlns="http://www.w3.org/2001/
XMLSchema"
     targetNamespace="http:// www.institution.org"
      xmlns:i="http://www.institution.org"
       attributeFormDefault="unqualified"
        elementFormDefault="unqualified">
<element name="institution">
<complexType>
<sequence>
<element name="name" type="string"/>
<element name="country" type="string"/>
</sequence>
<attribute name="ID" type="string"
     use="required"/>
</complexType>
</element>
</schema>
```

Figure 2. The first schema version of institutions (file Institution_V1.xsd)

```
<?xml version="1.0" encoding="UTF-8"?>
<t:thesis ID="PhD0611"
          xmlns:t="http://www.thesis.org"
          xmlns:foaf="http://xmlns.com/foaf/0.1/"
          xmlns:xsi="http://www.w3.org/2001/
                    XMLSchema-instance"
xsi:schemaLocation="http://www.thesis.org
                    Thesis_V1.xsd">
<title>Schema Versioning in XML Databases</title>
<student ID="Std9914">
<name>Zouhaier Brahmia</name>
<discipline>Computer Science</discipline>
</student>
<advisor ID="Adv8821">
<title>Professor</title>
<name>Rafik Bouaziz</name>
<discipline>Data Processing Systems</discipline>
</advisor>
<university ID="Univ7594">
<name>University of Sfax</name>
<country>Tunisia</country>
</university>
</t:thesis>
```

Figure 3. The first document version of theses (file Thesis_V1.xml)

Notice that the XML database designer has designed the XML schema of theses (Thesis_V1.xsd) by explicitly qualifying both (i) references to all components in the targetNamespace (http://www.thesis.org), and (ii) all components from the XML Schema namespace (http://

www.w3.org/2001/XMLSchema); he/she does not use a default namespace. Notice also that there are two other ways the designer could follow to design such a schema:

a) specifying the targetNamespace as the default namespace, and explicitly qualifying all components from the XML Schema namespace;

b) putting XML Schema as the default namespace, and explicitly qualifying all references to components in the targetNamespace.

However, he/she adopts the technique of case (b) for designing the schema of institutions (Institution_V1.xsd), which consists of making the XML Schema namespace the default namespace and explicitly qualifying all components from the corresponding targetNamespace (http://www.institution.org).

This example will be completed in Sec. 3.3 to show how changes to XML namespaces are specified and how their effects are propagated to both existing XML schemas and XML instances.

3. Our Approach for Managing XML Namespace Evolution

In this section, we introduce a principled approach to the definition of schema change operations devoted to changing XML namespaces, located in XML Schemas, and affecting underlying XML instance documents. We first present our design principles, then we describe schema change operations acting on XML namespaces. Last we show how these operations could be used by an XML database designer to perform graceful changes involving namespaces, in an XML database or in a repository of documents in XML format.

3.1. Design principles

The definition of schema change operations will obey the following principles and conventions:

1) All operations must have a well-formed and valid XML Schema Version (XSV) as input and produce a wellformed and valid XSV as output.

2) All operations need to work on an XSD file storing the XSV, whose name must be supplied as argument.

3) For all operations, arguments which are used to identify the object on which the operation works are in the first place of the argument list.

4) All operations must have values defined for all arguments; there are no empty places in the argument list.

The operations listed in the next subsection have been defined according to the design principles presented above.

3.2. Operations for Changing XML Namespaces

In this subsection, we propose a set of thirty-three operations for changing XML namespaces and consistently propagating XML schema change effects to linked XML schema and XML document instances. We list them in Table 1 and, for each operation, we provide a description of its parameters and of its operational semantics. These operations have been defined in order to satisfy requirements of XML database designers and XML schema developers, related to changes of XML namespaces. For that reason, we have tried to address all possible changes that could be done on any given XML namespace specified in an XML schema, based on (i) the study of the W3C document dealing with XML namespaces [15], and (ii) the common evolution needs of some XML-based application developers and designers (e.g., [13, 14, 22-26). Thus, we have identified the following types of operations:

a) adding, dropping, and renaming an XML namespace;

b) changing a target namespace;

c) transforming a default namespace into a non-default one (and *vice versa*);

d) exposing, and hiding an XML namespace for an element (for all elements, respectively) or for an attribute (for all attributes, respectively);

e) transforming a local element into a global element (and *vice versa*);

f) splitting an XML namespace into several new namespaces (at least two new namespaces);

g) merging a set of namespaces (at least two namespaces) into one XML namespace.

Furthermore, due to space limitations, we do not present in this work the effects of all schema change operations. We give only the effect of some of them. In the following, we choose to present only the effects of AddNamespace, ReplaceNamespace, and DefineTargetNamespace change operations.

The last operation in the table is not properly a schema change operation but can be used after a sequence of schema changes to propagate the changes to the underlying instances. Notice that we assume that propagation of changes is not automatic but must be explicitly requested by the DBA using the generalpurpose PropagateChangesTo operation, whose arguments are the names of XML instance files to which the changes have to be propagated. Such files are not necessarily all the XML instances conforming to the old schema versions, giving greater flexibility to the schema evolution process. During the execution of a transaction, we assume the system keeps track of the composition of changes that have to be propagated, so that (multiple) PropagateChangesTo operations can be executed at any time before the transaction commit. In case no PropagateChangesTo operations are executed, the new

Schema change operation	Semantics
AddNamespace (XSV.xsd, namespace, namespaceID)	It adds the XML namespace "namespace" with specified namespace identifier "namespaceID" to the new XML schema version (XSV.xsd) and to all its underlying XML documents.
DeleteNamespace (XSV.xsd, namespace)	It removes the XML namespace "namespace" from the new XML schema version (XSV.xsd) and from all its underlying XML documents.
ReplaceNamespace (XSV.xsd, namespace2, namespace3)	It replaces the namespace "namespace2" with the namespace "namespace3", in the new XML schema version (XSV.xsd), in all its underlying XML documents, and in all XML documents that are using (or referring to) elements and attributes belonging to the replaced XML namespace. Notice here that XSV.xsd is the XSD file on which the operation works, that can be either a definitional (if it contains a targetNamespace declaration) or a non-definitional XSD file As for the types of theother two arguments of the operation, only four valid combinations could be distinguished: 1) XSV.xsd is definitional, namespace2 is a new definitional XSD file, and namespace3 is a new URI possibly null. 2) XSV.xsd is definitional, namespace2 is null (i.e., the new definitional XSD file is void), and namespace3 is a new URI. 3) XSV.xsd is not definitional, namespace2 is the old URI to replace, and namespace3 is the new URI. 4) XSV.xsd is not definitional, namespace2 is the old URI to replace, and namespace3 is a new definitional XSD file.
DefineTargetNamespace(XSV.xsd, targetNamespace,targetNamespaceID)	It transforms a non-definitional XML schema version XSV.xsd into a definitional XSD file, by defining "targetNamespace" as the target namespace of XSV.xsd, with a possible target namespace identifier "targetNamespaceID", and by propagating such a definition to all XSD and XML files that are referencing XSV.xsd.
ChangeTargetNamespace(XSV.xsd, newTargetNamespace)	It sets the target namespace of the new XML schema version (XSV.xsd) to "newTargetNamespace". Notice here that newTargetNamespace must be a namespace that has an XSD file behind.
DropTargetNamespace(XSV.xsd)	It removes the target namespace of the new XML schema version (XSV.xsd) and from all its underlying XML documents.
TransformToDefaultNamespace(XSV.xsd, namespace)	It transforms the non-default XML namespace "namespace" into a default namespace, in the new XML schema version (XSV.xsd) and in all its underlying XML documents. If a namespace was previously defined as default, it should be contextually demoted to non default.
TransformToNonDefaultNamespa ce(XSV.xsd, namespace)	It transforms the default XML namespace "namespace" into a non-default namespace, in the new XML schema version (XSV.xsd) and in all its underlying XML documents.
ExposeNamespace (XSV.xsd, namespace)	It exposes the XML namespace "namespace" for all elements and attributes belonging to it, in the new XML schema version (XSV.xsd), in all its underlying XML documents, and in all XML documents that are using (or referring to) these elements and attributes.
ExposeNamespaceForAnElement (XSV.xsd, elementPath)	It exposes the XML namespace only for the element located at "elementPath" in the new XML schema version (XSV.xsd), in all its underlying XML documents, and in all XML documents that are using (or referring to) this element.
ExposeNamespaceForElements(XSV.xsd, namespace)	It exposes the XML namespace "namespace" only for all elements belonging to it, in the new XML schema version (XSV.xsd), in all its underlying XML documents, and in all XML documents that are using (or referring to) these elements.

ExposeNamespaceForAttributes(XSV.xsd, namespace)	It exposes the XML namespace “namespace” only for attributes belonging to it, in the new XML schema version (XSV.xsd), in all its underlying XML documents, and in all XML documents that are using (or referring to) these attributes.
ExposeAllNamespaces(XSV.xsd)	It exposes all XML namespaces specified in the new XML schema version (XSV.xsd), in all its underlying XML documents, and in all XML documents that are using (or referring to) elements and attributes belonging to these XML namespaces.
HideNamespace(XSV.xsd, namespace)	It hides (or localizes) the XML namespace “namespace” for all elements and attributes belonging to it, in the new XML schema version (XSV.xsd), in all its underlying XML documents, and in all XML documents that are using (or referring to) these elements and attributes.
HideNamespaceForElements(XSV.xsd, namespace)	It hides the XML namespace “namespace” only for elements belonging to it, in the new XML schema version (XSV.xsd), in all its underlying XML documents, and in all XML documents that are using (or referring to) these elements.
HideNamespaceForAnElement(XSV.xsd, elementPath)	It hides the XML namespace “namespace” only for the element located at “elementPath” in the new XML schema version (XSV.xsd), in all its underlying XML documents, and in all XML documents that are using (or referring to) this element.
HideNamespaceForAttributes (XSV.xsd, namespace)	It hides the XML namespace “namespace” only for attributes belonging to it, in the new XML schema version (XSV.xsd), in all its underlying XML documents, and in all XML documents that are using (or referring to) these attributes.
HideNamespaceForAnAttribute(XSV.xsd, attributePath)	It hides the XML namespace “namespace” only for the attribute located at “attributePath” in the new XML schema version (XSV.xsd), in all its underlying XML documents, and in all XML documents that are using (or referring to) this attribute.
HideAllNamespaces(XSV.xsd)	It hides all XML namespaces specified in the new XML schema version (XSV.xsd), in all its underlying XML documents, and in all XML documents that are using (or referring to) elements and attributes belonging to these XML namespaces.
TransformToGlobalElement(XSV.xsd, localElementPath)	It transforms a local element (located at localElementPath) to a global one, in the new XML schema version (XSV.xsd).
TransformToLocalElement (XSV.xsd, global Element Path)	It transforms a global element (located at globalElementPath) to a local one, in the new XML schema version (XSV.xsd).
TransformToGlobalAttribute(XSV.xsd, localAttributePath)	It allows transforming a local attribute (located at localAttributePath) to a global one, in the new XML schema version (XSV.xsd).
TransformToLocalAttribute(XSV.xsd, globalAttributePath)	It transforms a global attribute (located at globalAttributePath) to a local one, in the new XML schema version (XSV.xsd).
CreateNamespaceByExtraction(X SV.xsd, NewNamespace,sourceXSV.xsd,	It creates a new XML namespace “newNamespace” by extracting some XSD code (which starts at “selectionBeginningPath” and ends at “selectionEndPath”) from a source XML schema version (“sourceXSV.xsd”) and saving it as a new target namespace of the new XML schema version (“XSV.xsd”), with specified option (leave, delete, or link). The option parameter values are detailed as follows:

	1) leave: the sourceXSV.xsd is left unchanged after the extraction (i.e. the operation simply saves a copy of the selected subschema into the new target schema XSV.xsd); 2) delete: the selected subschema is deleted from the sourceXSV.xsd after the extraction (this option require possible propagation of modifications to the associated XML documents); 3) link: the selected subschema is substituted in the sourceXSV.xsd by: <xsd:include namespace="newNamespace" schemaLocation="XSV.xsd"/> after the extraction. This option leaves the sourceXSV.xsd globally (virtually) unchanged even if it has been split into two parts with the extraction.
MergeNamespaces(XSV.xsd, sourceNamespace,targetNamespace, targetElement,position)	It inserts a source XML namespace "sourceNamespace" into another target XML namespace "targetNamespace", at a specified position (i.e., before or after) with regard to a target element (located at "targetElementPath"). The "targetNamespace" parameter is defined as the target namespace of the new XML schema version "XSV.xsd". Notice here that sourceNamespace and targetNamespace must have been defined in a definitional XSD file.
AddImport(XSV.xsd,namespace, Location)	It adds an import clause with specified "namespace" and schema "schemaLocation" parameters, to the new XML schema version (XSV.xsd).
DeleteImport(XSV.xsd,namespace, schemaLocation)	It removes the import clause, having the specified "namespace" and "schemaLocation" parameters, from the new XML schema version (XSV.xsd).
ChangeImport (XSV.xsd,oldNamespace, oldSchemaLocation, newNamespace, newSchemaLocation)	It changes the namespace and/or the schemaLocation of the import clause having "oldNamespace" and "oldSchemaLocation" as parameters, in the new XML schema version (XSV.xsd).
AddInclude(XSV.xsd,namespace, schemaLocation)	It adds an include clause with specified "namespace" and "schemaLocation" parameters, to the new XML schema version (XSV.xsd).
DeleteInclude(XSV.xsd,namespace, schemaLocation)	It removes the include clause, having the specified "namespace" and "schemaLocation" parameters, from the new XML schema version (XSV.xsd).
ChangeInclude(XSV.xsd,namespace, schemaLocation)	It changes the namespace and/or the schemaLocation of the include clause having "oldNamespace" and "oldSchemaLocation" as parameters, in the new XML schema version (XSV.xsd).
PropagateChangesTo(inst1.xml, inst2.xml, inst3.xml, …)	It propagates the schema changes executed so far to the XML instance files specified as arguments.

Table 1. List of Schema Change Operations

schema version initially has no associated XML instances.

We describe in detail the functioning of the ReplaceNamespace operation, which has a different purpose depending on the combination of the types of its arguments.

In case XSV.xsd is a definitional XSD file, containing definitions for the http://NS1.org namespace (i.e., it contains targetNamespace="http://NS1.org"), then the ReplaceNamespace(XSV.xsd, ns2, ns3) operation can be used either to change the definitional XSD file to ns2, either to change the namespace name to ns3, or both. In particular, ReplaceNamescape(XSV.xsd, XSV.xsd, http://NS2.org), simply renames the namespace as http://NS2.org (the definitional file is unchanged), ReplaceNamescape(XSV.xsd, NS2.xsd, null) or ReplaceNamescape(XSV.xsd, NS2.xsd, http://NS1.org) replace the definitional XSD file with NS2.xsd without changing the namespace name, and ReplaceNamescape (XSV.xsd, null, http://NS2.org) demotes the XSV.xsd file

to non-definitional (the new definitional XSD file is void) and the namespace http://NS1.org is replaced with http://NS2.org.

In case XSV.xsd is not a definitional XSD file (i.e., it does not contain a targetNamespace declaration), ReplaceNamespace ReplaceNamespace(XSV.xsd, ns2, ns3) allows to replace the namespace with name ns2 (e.g., ns2 is the URI http://NS1.org; notice that more than one namespace can be declared in XSV.xsd) with the namespace ns3. If ns3 is a namespace name (e.g., ns2 is the URI http://NS2.org), it simply replaces ns2 with ns3 (i.e., http://NS1.org with http://NS2.org). If ns3 is a definitional XSD file, say NS2.xsd containing the targetNamespace="http://NS2.org" declaration, ns2 in XSV.xsd is substituted by URI http://NS2.org and an element <xsd:include namespace="http://NS2.org" schemaLocation="NS2.xsd"/> is also added.

Furthermore, we highlight the differences between the uses of AddNamespace and DefineTarget Name space operations by means of application examples. In order to show the effect of the AddNamespace and DefineTargetNamespace operations, let us assume that XSV.xsd is not a definitional XSD file (i.e., it doesn't contain targetNamespace) and its contents are, for instance, as follows:

```
<schema xmlns="http://www.w3.org/2001/XMLSchema"
xmlns:ns1="http://NS1.org">
…
</schema>
```

then AddNamespace(XSV.xsd, http://NS2.org, ns2) produces:

```
<schema xmlns="http://www.w3.org/2001/XMLSchema"
xmlns:ns1="http://NS1.org"
xmlns:ns2="http://NS2.org">
…
</schema>
```

On the other hand, starting from the same initial XSV.xsd file, DefineTargetNamespace(XSV.xsd, http://NS2.org, null) produces:

```
<schema xmlns="http://www.w3.org/2001/XMLSchema"
xmlns:ns1="http://NS1.org"
targetNamespace="http://NS2.org">
…
</schema>
```

and can be propagated to XSD/XML files referencing XSV.xsd, for example, by changing:

```
<xsd:include schemaLocation="XSV.xsd"/>
```

to:

```
<xsd:import namespace="http://NS2.org"
schemaLocation="XSV.xsd"/>
```

When also a namespace identifier is specified, as in DefineTargetNamespace(XSV.xsd, http://NS2.org, ns2), the effects on XSV.xsd are as follows:

```
<schema xmlns="http://www.w3.org/2001/XMLSchema"
xmlns:ns1="http://NS1.org"
xmlns:ns2="http://NS2.org"
targetNamespace="http://NS2.org">
…
</schema>
```

Furthermore, as far as the effect of the ReplaceNamespace operation is concerned, assume the contents of XSV.xsd are as follows:

```
<schema xmlns="http://www.w3.org/2001/XMLSchema"
targetNamespace="http://NS1.org"
xmlns:ns2="http://NS2.org"
xmlns:ns3="http://NS3.org">
…
</schema>
```

then ReplaceNamespace(XSV.xsd, http://NS1.org, http://NS3.org) produces:

```
<schema xmlns="http://www.w3.org/2001/XMLSchema"
targetNamespace="http://NS3.org"
xmlns:ns2="http://NS2.org"
xmlns:ns3="http://NS3.org">
…
</schema>
```

and can be propagated to XSD/XML files referencing XSV.xsd, for example, by changing:

```
<xsd:include namespace="http://NS1.org"
schemaLocation="XSV.xsd"/>
and
<xsd:import namespace="http://NS1.org"
schemaLocation="XSV.xsd"/>
to:
<xsd:include namespace="http://NS3.org"
schemaLocation="XSV.xsd"/>
and
<xsd:import namespace="http://NS3.org"
schemaLocation="XSV.xsd"/>
```

respectively.

3.3. Running Example Reprise

In order to illustrate the functioning of the proposed approach for managing XML namespace evolution, we continue in this subsection our example of Sec. 2.2, in order to show how XML namespace changes are dealt with in our approach. We suppose that the XML database designer has to apply, at different instants, four schema changes dealing with XML namespaces. We present these changes and show their effects at both schema and instance levels.

Schema Change no. 1. Suppose that the XML database

designer wants to change the target namespace of institutions so that they will have the same target namespace as theses (i.e., http://www.thesis.org). Consequently, the new schema version of institutions (i.e., Institution_V2.xsd, see Fig. 5) has to be recalled, in the new schema version of theses (i.e., Thesis_V2.xsd, see Fig. 4), via xsd:include rather than via xsd:import. Furthermore, a new version of the XML document corresponding to theses (i.e., Thesis_V2.xml, see Fig. 6) has to be generated from the previous one (i.e., Thesis_V1.xml, see Fig. 3).

The sequence of operations, which must be specified by the XML database designer in order to perform this namespace change, consists of the following schema change transaction:

Begin Transaction

(i) CreateNewXMLSchemaVersion(Institution_V2.xsd, Institution_V1.xsd)
(ii) CreateNewXMLSchemaVersion(Thesis_V2.xsd, Thesis_V1.xsd)
(iii) ChangeTargetNamespace(Institution_V2.xsd, http://www.thesis.org)
(iv)DeleteImport(ThesisV2.xsd, http://www.institution.org, Institution_V1.xsd)
(v) AddInclude(Thesis_V2.xsd, http://www.thesis.org, Institution_V2.xsd)
(vi) CreateNewXMLDocumentVersion(Thesis_V1.xml, Thesis_V2.xml)
(vii) PropagateChangesTo(Thesis_V2.xml)

Commit

Notice here that we assume that the operation "CreateNew XMLSchemaVersion(NewSV.xsd, ExistingSV.xsd)" ("CreateNewXMLDocumentVersion(NewDV.xml, ExistingDV.xml"), respectively) initially generates a new XML schema version "NewSV.xsd" (a new XML document version "NewDV.xml", respectively) from an existing one "ExistingSV.xsd" ("ExistingDV.xml"), respectively) as a copy of it. After that, NewSV.xsd (NewDV.xml, respectively) is updated by the other operations of the schema change transaction, in order to obtain, in the end, a new XML schema version (XML document version, respectively) that is actually different from the initial one according to the schema change specifications (the schema change propagation, respectively).

Schema Change no. 2. Suppose that the XML database designer wants to hide (localize) XML namespaces of theses and to expose XML namespaces of students, advisors and institutions. Hence, he/she updates (i) the XML Schema document Thesis_V2.xsd by setting the elementFormDefault and the attributeFormDefault attributes of the <schema/> element to "unqualified", and (ii) the XML Schema document Institution_V2.xsd by setting the element Form Default and the attribute Form default attributes of the <schema/> element to "qualified".

```
<?xml version="1.0" encoding="UTF-8"?>
<xsd:schema
      xmlns:xsd="http://www.w3.org/2001/XMLSchema"
       targetNamespace="http://www.thesis.org"
        xmlns:t="http://www.thesis.org"
        xmlns:foaf="http://xmlns.com/foaf/0.1/"
       attributeFormDefault="unqualified"
      elementFormDefault="unqualified">
<xsd:include namespace="http://www.thesis.com"
schemaLocation="Institution_V2.xsd"/>
<xsd:element name="thesis">
<xsd:complexType>
<xsd:sequence>
<xsd:element name="title" type="xsd:string"/>
<xsd:element name="student"
       type="foaf:Person"/>
<xsd:element name="advisor"
       type="foaf:Person"/>
<xsd:element name="university"
       type="t:institution"/>
</xsd:sequence>
<xsd:attribute name="ID" type="xsd:string"
       use="required"/>
</xsd:complexType>
</xsd:element>
</xsd:schema>
```

Figure 4. The second schema version of theses (file Thesis_V2.xsd)

```
<?xml version="1.0" encoding="UTF-8"?>
<schema xmlns="http://www.w3.org/2001/XMLSchema"
          targetNamespace="http://www.thesis.org"
          xmlns:t="http://www.thesis.org"
          attributeFormDefault="unqualified"
          elementFormDefault="unqualified">
<element name="institution">
<complexType>
<sequence>
<element name="name" type="string"/>
<element name="country" type="string"/>
</sequence>
<attribute name="ID" type="string"
          use="required"/>
</complexType>
</element>
</schema>
```

Figure 5. The second schema version of institutions (file Institution_V2.xsd)

A new version of each one of these updated schemas is then created (i.e., Thesis_V3.xsd as shown in Fig. 7, and Institution_V3.xsd as shown in Fig. 8), and a new XML document (Thesis_V3.xml, see Fig. 9) is also generated from Thesis_V2.xml as a new version.

```
<?xml version="1.0" encoding="UTF-8"?>
<t:thesis ID="PhD0611"
        xmlns:t="http://www.thesis.org"
        xmlns:foaf="http://xmlns.com/foaf/0.1/"
        xmlns:xsi="http://www.w3.org/2001/
        XMLSchema-instance"
        xsi:schemaLocation="http://www.thesis.org
        Thesis_V2.xsd">
<title>Schema Versioning in XML Databases</title>
<student ID="Std9914">
        <name>Zouhaier Brahmia</name>
        <discipline>Computer Science</discipline>
</student>
<advisor ID="Adv8821">
        <title>Professor</title>
        <name>Rafik Bouaziz</name>
        <discipline>Data Processing Systems</
discipline>
</advisor>
<university ID="Univ7594">
        <name>University of Sfax</name>
        <country>Tunisia</country>
</university> </t:thesis>
```

Figure 6. The second document version of theses (file Thesis_V2.xml)

The sequence of operations, which must be specified by the XML database designer in order to perform this namespace change, is listed in the following schema change transaction:

Begin Transaction

(i) CreateNewXMLSchemaVersion(Thesis_V3.xsd, Thesis_V2.xsd)
(ii) CreateNewXMLSchemaVersion(Institution_V3.xsd, Institution_V2.xsd)
(iii) HideNamespace(Thesis_V3.xsd, http://www.thesis.org)
(iv) ExposeNamespace(Thesis_V3.xsd, http://xmlns.com/foaf/0.1/)
(v) ExposeNamespace(Thesis_V3.xsd, http://www.institution.com)
(vi) ExposeAllNamespaces(Institution_V3.xsd)
(vii) CreateNewXMLDocumentVersion(Thesis_V2.xml, Thesis_V3.xml)
(viii) PropagateChangesTo(Thesis_V3.xml)

Commit

Schema Change no. 3. Suppose that the XML database designer wants to declare both the attribute "ID" and all subelements of the <thesis/> element (i.e., <title/>, <student/>, <advisor/>, and <university/>) as global ones (i.e., children of the <schema/> element).

After updating the XML Schema document Thesis_V3.xsd, the designer obtains a new schema version Thesis_V4.xsd as shown in Fig. 10. A new version of the underlying XML document (Thesis_V4.xml, see Fig. 11) is also automatically generated from Thesis_V3.xml.

The sequence of operations, which must be specified by the XML database designer in order to perform this namespace change, makes up the following schema change transaction:

```
<?xml version="1.0" encoding="UTF-8"?>
<xsd:schema
    xmlns:xsd="http://www.w3.org/2001/XMLSchema"
    targetNamespace="http://www.thesis.org"
     xmlns="http://www.thesis.org"
     xmlns:foaf="http://xmlns.com/foaf/0.1/"
      attributeFormDefault="unqualified"
      elementFormDefault="unqualified">
<xsd:include namespace="http://www.thesis.com"
   schemaLocation="Institution_V3.xsd"/>
<xsd:element name="thesis">
<xsd:complexType>
<xsd:sequence>
<xsd:element name="title" type="xsd:string"/>
<xsd:element name="student"
    type="foaf:Person"/>
<xsd:element name="advisor"
    type="foaf:Person"/>
<xsd:element name="university"
     type="institution"/>
</xsd:sequence>
<xsd:attribute name="ID" type="xsd:string"
    use="required"/>
</xsd:complexType>
</xsd:element>
</xsd:schema>
```

Figure 7. The third schema version of theses (file Thesis_V3.xsd)

```
<?xml version="1.0" encoding="UTF-8"?>
<schema xmlns="http://www.w3.org/2001/XMLSchema"
        targetNamespace="http://www.thesis.org"
        xmlns:t="http://www.thesis.org"
        attributeFormDefault="qualified"
        elementFormDefault="qualified">
<element name="institution">
<complexType>
<sequence>
<element name="name" type="string"/>
<element name="country" type="string"/>
</sequence>
<attribute name="ID" type="string"
        use="required"/>
</complexType>
</element> </schema>
```

Figure 8. The third schema version of institutions (file Institution_V3.xsd)

Begin Transaction
(i) CreateNewXMLSchemaVersion(Thesis_V4.xsd, Thesis_V3.xsd)
(ii) TransformToGlobalElement(Thesis_V4.xsd, "//xsd:element[@name='title']")
(iii) TransformToGlobalElement(Thesis_V4.xsd, "//xsd:element[@name='student']")
(iv) TransformToGlobalElement(Thesis_V4.xsd, "//xsd:element[@name='advisor']")
(v) TransformToGlobalElement(Thesis_V4.xsd, "//xsd:element[@name='university']")
(vi) TransformToGlobalAttribute(Thesis_V4.xsd, "//xsd:attribute[@name='ID']")
(vii) CreateNewXMLDocumentVersion(Thesis_V3.xml, Thesis_V4.xml)
(viii) PropagateChangesTo(Thesis_V4.xml)

Commit

Notice here that the two documents Thesis_V4.xml (seeFig. 11) and Thesis_V3.xml (see Fig. 9), except the reference to a different schema (i.e., Thesis_V4.xsd and Thesis_V3.xsd, respectively), have exactly the same contents although their XML schemas are different.

```
<?xml version="1.0" encoding="UTF-8"?>
<t:thesis ID="PhD0611"
        xmlns:t="http://www.thesis.org"
        xmlns:foaf="http://xmlns.com/foaf/0.1/"
        xmlns:xsi="http://www.w3.org/2001/
        XMLSchema-instance"
        xsi:schemaLocation="http://www.thesis.org
        Thesis_V3.xsd">
<title>Schema Versioning in XML Databases</title>
<student foaf:ID="Std9914">
<foaf:name>Zouhaier Brahmia</foaf:name>
<foaf:discipline>Computer Science
</foaf:discipline>
</student>
<advisor foaf:ID="Adv8821">
<foaf:title>Professor</foaf:title>
<foaf:name>Rafik Bouaziz</foaf:name>
<foaf:discipline>Data Processing Systems
</foaf:discipline>
</advisor>
<university t:ID="Univ7594">
<t:name>University of Sfax</t:name>
<t:country>Tunisia</t:country>
</university>
</t:thesis>
```

Figure 9. The third document version of theses (file Thesis_V3.xml)

Notice also that if there were (different) local elements with the same name which, thus, would conflict at the global level, the schema change could be either (i) automatically rejected by the system, or (ii) performed

```
<?xml version="1.0" encoding="UTF-8"?>
<xsd:schema
      xmlns:xsd="http://www.w3.org/2001/XMLSchema"
       targetNamespace="http://www.thesis.org"
        xmlns:t="http://www.thesis.org"
        xmlns:foaf="http://xmlns.com/foaf/0.1/"
       attributeFormDefault="unqualified"
       elementFormDefault="unqualified">
<xsd:include
       namespace="http://www.institution.com"
       schemaLocation="Institution_V3.xsd"/>
<xsd:element name="title" type="xsd:string"/>
<xsd:element name="student" type="foaf:Person"/
>
<xsd:element name="advisor" type="foaf:Person"/
>
<xsd:element name="university"
       type="t:institution"/>
<xsd:attribute name="ID" type="xsd:string"/>
<xsd:element name="thesis">
<xsd:complexType>
<xsd:sequence>
        <xsd:element ref="t:title"/>
       <xsd:element ref="t:student"/>
       <xsd:element ref="t:advisor"/>
       <xsd:element ref="t:university"/>
</xsd:sequence>
<xsd:attribute ref="t:ID" use="required"/>
</xsd:complexType>
</xsd:element>
</xsd:schema>
```

Figure 10. The fourth schema version of theses (file Thesis_V4.xsd)

after asking the designer to provide new names for the corresponding local elements.

Schema Change no. 4. Suppose that the XML database designer wants to rename the namespace of theses (i.e., http://www.thesis.org) to "http://www.NationalPhDtheses.org" without changing its definition.

After updating the XML Schema documents Thesis_V4.xsd, the designer obtains a new XML schema version, Thesis_V5.xsd, as shown in Fig. 12. A new version of the underlying XML document (Thesis_V5.xml, see Fig. 13) is also automatically generated from Thesis_V4.xml.

The sequence of operations, which must be specified by the XML database designer in order to perform this namespace change, consists of the following schema change transaction:

Begin Transaction

(i) CreateNewXMLSchemaVersion(Thesis_V5.xsd, Thesis_V4.xsd)
(ii) ReplaceNamespace(Thesis_V5.xsd, http://www.thesis.org, http://www.NationalPhDtheses.org)
(iii) CreateNewXMLDocumentVersion(Thesis_V4.xml, Thesis_V5.xml)
(iv) PropagateChangesTo(Thesis_V5.xml)

Commit

4. Related Work Discussion

In the literature of XML databases, there are several works that have dealt with XML schema changes in an environment which supports either XML schema evolution (e.g., [24, 27-33]) or XML schema versioning (e.g., [11-14, 34-36]). They studied changes acting on XSD elements, XSD attributes, or XML subschema. However, they did not study how changes to XML namespaces could be consistently performed on XML schema documents and safely propagated to XML document instances.

```
<?xml version="1.0" encoding="UTF-8"?>
<t:thesis ID="PhD0611"
        xmlns:t="http://www.thesis.org"
        xmlns:foaf="http://xmlns.com/foaf/0.1/"
        xmlns:xsi="http://www.w3.org/2001/
        XMLSchema-instance"
        xsi:schemaLocation="http://www.thesis.org
        Thesis_V4.xsd">
<title>Schema Versioning in XML Databases</title>
<student foaf:ID="Std9914">
<foaf:name>Zouhaier Brahmia</foaf:name>
<foaf:discipline>Computer Science
</foaf:discipline>
</student>
<advisor foaf:ID="Adv8821">
<foaf:title>Professor</foaf:title>
<foaf:name>Rafik Bouaziz</foaf:name>
<foaf:discipline>Data Processing Systems
</foaf:discipline>
</advisor>
<university t:ID="Univ7594">
<t:name>University of Sfax</t:name>
<t:country>Tunisia</t:country>
</university>
</t:thesis>
```

Figure 11. The fourth document version of theses (file Thesis_V4.xml)

In an excellent recent survey, Faisal and Sarwar [23] studied the temporal and versioning aspects in XML document management. But, there is no work which has studied changes to XML namespaces.

Nösinger et al. [33] propose ELaX, a language that allows specifying complex XML schema updates, in a formal way. Nevertheless, it does not support operations for changing XML namespaces.

```
<?xml version="1.0" encoding="UTF-8"?>
<xsd:schema
        xmlns:xsd="http://www.w3.org/2001/
        XMLSchema"targetNamespace="http://
        www.NationalPhDtheses.org"
xmlns:t="http://www.NationalPhDtheses.org"
        xmlns:foaf="http://xmlns.com/foaf/0.1/"
        attributeFormDefault="unqualified"
        elementFormDefault="unqualified">
<xsd:include
        namespace="http://www.institution.com"
        schemaLocation="Institution_V3.xsd"/>
<xsd:element name="title" type="xsd:string"/>
<xsd:element name="student" type="foaf:Person"/>
<xsd:element name="advisor" type="foaf:Person"/>
<xsd:element name="university"
        type="t:institution"/>
<xsd:attribute name="ID" type="xsd:string"/>
<xsd:element name="thesis">
<xsd:complexType>
<xsd:sequence>
<xsd:element ref="t:title"/>
<xsd:element ref="t:student"/>
<xsd:element ref="t:advisor"/>
<xsd:element ref="t:university"/>
</xsd:sequence>
<xsd:attribute ref="t:ID" use="required"/>
</xsd:complexType>
</xsd:element>
</xsd:schema>
```

Figure 12. The fifth schema version of theses (file Thesis_V5.xsd)

In [37], the authors deal with merging of XML schema versions and resolving conflicts that result from such operation, without any information on namespace merging or conflicts.

Brahmia et al. [38] propose a set of high-level operations for changing conventional and temporal schema the TXSchema framework [4, 11, 34, 36], but they do not provide any operation for managing changes to XML namespaces.

In [31], a framework for changing XML schemas has been presented. It allows the designer to perform schema updates at conceptual level (i.e., on UML diagrams) and propagates such updates to involved XSD and XML documents.

Some authors have proposed approaches for evolving XML schemas, which are based on the Model-Driven

Development, like in [32]; some other authors were based on the Model- Driven Architecture, like in [12] and [39].

Kwietniewski et al. [29] and Hasegawa et al. [40] have dealt with transformations of XML documents when their XML schemas evolve. In [29], the authors propose an approach that generates XSLT files that will be executedon the corresponding XML document in order to make it valid with respect to the new XML schema. The authors of [40] focus on transforming XPath expressions in existing XML documents so that they conform to their updated schema.

```
<?xml version="1.0" encoding="UTF-8"?>
<t:thesis ID="PhD0611"
        xmlns:t="http://www.NationalPhDtheses.org"
        xmlns:xsi="http://www.w3.org/2001/
                XMLSchema-instance"
        xsi:schemaLocation="http://
www.NationalPhDtheses.org Thesis_V5.xsd">
<title>Schema Versioning in XML Databases</title>
<student foaf:ID="Std9914">
<foaf:name>Zouhaier Brahmia</foaf:name>
<foaf:discipline>Computer
Science</foaf:discipline>
</student>
<advisor foaf:ID="Adv8821">
<foaf:title>Professor</foaf:title>
<foaf:name>Rafik Bouaziz</foaf:name>
<foaf:discipline>Data Processing Systems
</foaf:discipline>
</advisor>
<university t:ID="Univ7594">
<t:name>University of Sfax</t:name>
<t:country>Tunisia</t:country>
</university>
</t:thesis>
```

Figure 13. The fifth document version of theses (file Thesis_V5.xml)

XML schema versioning has been widely studied in the temporal XML context (e.g., [11-13], [34-38]). However, (temporal) XML namespace versioning has never been studied.

Four research prototypes, CoDEX [27], X-Evolution [28], EXup [30], and eXolution [24], were proposed for managing XML Schema evolution. However, they do not support changing XML namespaces and propagating their effects to underlying XML documents.

Mainstream native XML DBMSs (like EMC xDB, eXist, and Sedna) and XML management tools (like Altova's XMLSpy, XMLmind's XML Editor, and Oxygen's XML Editor) support some XML schema change operations, like adding/dropping/renaming an XSD element/attribute, as mentioned in [9] and [41]. Nevertheless, none of them supports changes to XML namespaces (specified in XML schema documents).

As far as mainstream commercial DBMSs (e.g., Oracle, Microsoft SQL Server, and IBM DB2) are concerned, although they provide a limited support for managing (some) XML schema changes, as shown in [9, 23, 41], they do not provide any explicit support for changing XML namespaces and propagating their effects to underlying XML documents.

5. Conclusion

In this work, we have studied XML namespace changes and their effects on XML instances under schema versioning. We proposed a systematic approach that is based on a large set of operations allowing XML database designers and administrators to gracefully manage changes to XML namespaces. Propagation of namespace changes to linked XML schema and XML instance documents is automatically effected by the proposed schema change operations. We illustrated our proposal through a running example. To the best of our knowledge, we are the first to study the namespace change problem (and in an XML schema versioning context).

In the near future, we will start developing a prototype system, in order to show the technical feasibility of our approach. We intend to provide a graphical tool that allows the designer to choose the desired XML schema file(s) and to perform namespace changes via a graphical interface.

Although in our current work we consider only simple XML documents (i.e., XML documents with only nested elements, attributes and textual contents, without other kinds of features or constraints), we plan to focus on more complicated aspects (e.g., XML integrity constraints) related to this topic in our future work.

Furthermore, we will also extend our work to dealing with changes of high-level design patterns: we intend to study how to transform an XML document formatted according to a given design pattern (e.g., Garden of Eden) into another XML document formatted according to another design pattern (e.g., Venetian Blind).

References

[1] Bourret, R (2005). XML and Databases, September. <http://www.rpbourret.com/xml/XMLAndDatabases.htm>

[2] Chaudhri, A. Zicari, R., Rashid, A (2003). *XML Data Management: Native XML and XML Enabled DataBase Systems*. Boston, MA, USA: Addison-Wesley Longman Publishing Co., Inc.

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

[4] Currim, F., Currim, S., Dyreson, C.E., et al. (2004). A Tale of Two Schemas: Creating a Temporal XML Schema

from a Snapshot Schema with TXSCHEMA. *In:* Proc. of the 9th International Conference on Extending Database Technology (EDBT 2004), Crete, Greece, 14-18 March, p. 348-365.

[5] Etzion, O., Jajodia, S., Sripada, S. (Eds.) (1998).*Temporal Databases — Research and Practice*. Berlin, Germany: Springer-Verlag. LNCS No. 1399.

[6] Grandi, F (2015). Temporal databases. In: Khosrow-Pour, M. (Ed.), *Encyclopedia of Information Science and Technology (3rd edition)*. Hershey, PA, USA: IGI Global, p. 1914–1922.

[7] Grandi, F., Mandreoli, F (2003). A Formal Model for Temporal Schema Versioning in Object-Oriented Databases. *Data & Knowledge Engineering* 46 (2) 123–167.

[8] Roddick, J.F (2009). Schema Versioning. *In*: Liu, L., Özsu, M.T. (Eds.), *Encyclopedia of Database Systems*. Heidelberg, Germany: Springer-Verlag, p. 2499-2502.

[9] Brahmia, Z., Grandi, F., Oliboni, B., et al. (2015). Schema Versioning. In: Khosrow-Pour, M. (Ed.), *Encyclopedia of Information Science and Technology (3rd edition)*. Hershey, PA, USA: IGI Global, p. 7651-7661.

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

[11] Snodgrass, R.T., Dyreson, C.E., Currim, F., et al. (2008). Validating Quicksand: Temporal Schema Versioning in TXSCHEMA. *Data & Knowledge Engineering* 65 (2) 223- 242.

[12] Malý, J., Mlýnková, I., Neaský, M (2011). XML Data Transformations as Schema Evolves. In: *Proc. of the 15th International Conference on Advances in Databases and Information Systems (ADBIS 2011)*, Vienna, Austria, 20-23 September, p. 375-388.

[13] Brahmia, Z., Grandi, F., Oliboni, B., et al. (2014). Schema Change Operations for Full Support of Schema Versioning in the TXSCHEMA Framework. *International Journal of Information Technology and Web Engineering* 9 (2) 20-46.

[14] Baqasah, A., Pardede, E., Rahayu, J.W (2015). Maintaining Schema Versions Compatibility in Cloud Applications Collaborative Framework. *World Wide Web* 18 (6) 1541-1577.

[15] Namespaces in XML 1.0 (Third Edition), W3C Recommendation, 8 December 2009. <http://www.w3.org/TR/2009/REC-xml-names-20091208/>

[16] Bourret, R (2000). Namespace Myths Exploded. <http://www.rpbourret.com/xml/NamespaceMyths.htm>

[17] Bourret, R (2009). XML Namespaces FAQ, January. <http://www.rpbourret.com/xml/NamespacesFAQ.htm>

[18] Berners-Lee, T., Fielding, R., Masinter, L. (Eds.) (2005). RFC 3986: Uniform Resource Identifier (URI): Generic Syntax, IETF (Internet Engineering Task Force), January 2005. <http://www.rfc-editor.org/rfc/rfc3986.txt>

[19] Brahmia, Z., Grandi, F., Bouaziz, R (2016). Changes to XML Namespaces in XML Schemas and their Effects on Associated XML Documents under Schema Versioning. *In:* Proceedings of the 11th International Conference on Digital Information Management (ICDIM 2016), Porto, Portugal, 19-21 September, p. 43-50.

[20] Grandi, F (2004). Introducing an Annotated Bibliography on Temporal and Evolution Aspects in the World Wide Web. *SIGMOD Record* 33 (2) 84-86.

[21] Guerrini, G., Mesiti, M (2009). XML Schema Evolution and Versioning: Current Approaches and Future Trends. *In*: Pardede, E. (Ed.), *Open and Novel Issues in XML Database Applications: Future Directions and Advanced Technologies*. Hershey, PA, USA: Information Science Reference – IGI Global, p. 66-87.

[22] Colazzo, D., Guerrini, G., Mesiti, M., et al. (2010). 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*. Hershey, PA, USA: Information Science Reference – IGI Global, p. 361-384.

[23] Faisal, S., Sarwar, M (2014). Temporal and multiversioned XML documents: A survey. *Information Processing and Management* 50 (1) 113–131.

[24] Klímek, J., Malý, J., Ne aský, M., et al. (2015). eXolutio: Methodology for Design and Evolution of XML Schemas using Conceptual Modeling. *Informatica* 26(3) 453-472.

[25] Curino, C., Moon, H.J., Tanca, L., et al. (2008). Schema Evolution in Wikipedia: toward a Web Information System Benchmark. *In*: Proc. of the 10th International Conference on Enterprise Information Systems (ICEIS 2008), Barcelona, Spain, 13-16 June, Volume DISI, p. 323-332.

[26] Curino, C., Moon, H.J., Deutsch, A., et al. (2013). Automating the database schema evolution process. *The VLDB Journal* 22(1) 73-98.

[27] Klettke, M (2007). Conceptual XML Schema Evolution - the CoDEX Approach for Design and Redesign. *In:* Workshop Proceedings of Datenbanksysteme in Business, Technologie und Web (BTW 2007), Aachen, Germany, 5-9 March, p. 53-63.

[28] Guerrini, G., Mesiti, M (2008). X-Evolution: A Comprehensive Approach for XML Schema Evolution. *In:* Proceedings of the 3rd International DEXA Workshop on XML Data Management Tools and Techniques (XANTEC'08), Turin, Italy, 1-5 September, p. 251-255.

[29] Kwietniewski, M., Gryz, J., Hazlewood, S., et al. (2010). Transforming XML Documents as Schemas Evolve. *Proceedings of the VLDB Endowment (PVLDB)* 3 (2) 1577-1580.

[30] Cavalieri, F., Guerrini, G., Mesiti, M (2011). Updating XML Schemas and Associated Documents through EXup.

In: Proceedings of the 27th International Conference on Data Engineering (ICDE 2011), Hannover, Germany, 11-16 April, p. 1320-1323.

[31] Domínguez, E., Lloret, J., Pérez, B., et al. (2011). Evolution of XML Schemas and documents fromstereotyped UML class models: A traceable approach. *Information & Software Technology* 53 (1) 34-50.

[32] Ne aský, M., Klímek, J., Malý, J., et al. (2012). Evolution and change management of XML-based systems. *Journal of Systems and Software* 85 (3) 683-707.

[33] Nösinger, T., Klettke, M., Heuer, A (2013). XML Schema Transformations - The ELaX Approach. *In*: Proceedings of the 24th International Conference Database and Expert Systems Applications (DEXA 2013), Prague, Czech Republic, 26-30 August, Part I, p. 293-302.

[34] Dyreson, C.E., Snodgrass, R.T., Currim, F., et al. (2006). Validating Quicksand: Schema Versioning in TXSCHEMA. *In*: Proceedings of the 3rd International ICDE Workshop on XML Schema and Data Management *(XSDM'06)*, Atlanta, GA, USA, 3-7 April, paper 82.

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

[36] Currim, F., Currim, S., Dyreson, C.E., et al. (2009). TXSCHEMA: Support for Data- and Schema-Versioned XML Documents. Technical Report TR-91, TimeCenter, September.<http://timecentre.cs.aau.dk/TimeCenter Publications/TR- 91.pdf>

[37] Baqasah, A., Pardede, E., Rahayu, J.W (2014). A New Approach for Meaningful XML Schema Merging. *In:* Proceedings of the 16th International Conference on Information Integration and Web-based Applications & Services (iiWAS 2014), Hanoi, Vietnam, 4-6 December, p. 430-439.

[38] Brahmia, Z., Grandi, F., Oliboni, B., et al. (2014). Highlevel Operations for Creation and Maintenance of Temporal and Conventional Schema in the TXSCHEMA Framework. *In:* Proceedings of the 21st International Symposium on Temporal Representation and Reasoning (TIME'2014), Verona, Italy, 8-10 September, p. 101- 110.

[39] Klímek, J., Malý, J., Mlýnková, I., et al. (2012). eXolutio: Tool for XML Schema and Data Management. *In*: Proceedings of the 12th Annual International Workshop on DAtabases, TExts, Specifications and Objects (DATESO 2012), Zernov, Rovensko pod Troskami, Czech Republic, 18-20 April, p. 69-80.

[40] Hasegawa, K., Ikeda, K., Suzuki, N (2013). An Algorithm for Transforming XPath Expressions According to Schema Evolution. *In:* Proceedings of the International Workshop on Document Changes: Modeling, Detection, Storage and Visualization (DChanges 2013), Florence, Italy, 10 September, paper 4.

[41] Brahmia, Z., Grandi, F., Oliboni, B., et al. (2015). Schema Evolution. *In:* Khosrow-Pour, M. (Ed.), Encyclopedia of Information Science and Technology (3rd edition). Hershey, PA, USA: IGI Global, p. 7641-7650.