Updating XML Schemas and Associated Documents through EXup

Federico Cavalieri[1], Giovanna Guerrini[1], Marco Mesiti[2]

[1]*DISI, University of Genova, Italy*
{cavalieri, guerrini}@disi.unige.it

[2]*DiCo, University of Milano, Italy*
mesiti@dico.unimi.it

Abstract— **Data on the Web mostly are in XML format and the need often arises to update their structure, commonly described by an XML Schema. When a schema is modified the effects of the modification on documents need to be faced. XSUpdate is a language that allows to easily identify parts of an XML Schema, apply a modification primitive on them and finally define an adaptation for associated documents, while E$\mathcal{X}$up is the corresponding engine for processing schema modification and document adaptation statements. Purpose of this demonstration is to provide an overview of the facilities of the XSUpdate language and of the E$\mathcal{X}$up system.**

I. Motivations

In the last few years we have observed a proliferation of approaches for querying and updating XML documents both from the research community and the big DBMS companies. Suitable languages have been provided for both querying and updating, the SQL standard has been extended to query and publish XML data, vendors have enhanced their DBMSs to support XML as a native type.

On the contrary, updates on XML schemas have received a limited attention despite the great impact they may have in the database organization and in many application fields. The need to update the schema may arise for several reasons, ranging to reflecting a change in the application domain, to a restructuring of the business alliance sharing the data, to revisions on recently developed and still unstable domain-specific data representation standards. Schema updates may have different impacts in data management in the context of *schema versioning*, where the updated schema gives rise to a new schema version, and *schema evolution*, where the updated schema replaces the previous one.

Commercial DBMSs, like Oracle 11g, Tamino, DB2 v.9, provide different support for schema versioning but the possibilities to handle schema evolution is quite limited and mainly rely on the use of ad-hoc routines (like the Oracle `copyEvolve` and `inPlaceEvolve` functions). There is therefore the need to identify a language for updating schemas that is easy to use, whose semantics is well specified, and that copes with all the effects of schema updates on related documents. Specifically, in a schema evolution context, *old* documents may need to be (incrementally) revalidated against the new schema, and, if they are not valid anymore, *adapted* to the new schema.

The XQuery Update (XQU) facilities [4] can be exploited to face the issues of schema updates. However, the specification of updates on a schema results in quite verbose expressions, that are not easy to understand and error-prone. Moreover, the management of effects on related documents is completely decoupled from the schema modification and does not take advantage of knowledge of the update occurred on the schema.

In this demo paper we show the potential of XSUpdate [1], a language that we have developed for expressing modifications on XML schemas and for supporting incremental revalidation, as well as automatic and user-defined adaptation of documents associated with a schema. The capabilities of the language will be demonstrated by means of the E$\mathcal{X}$up system [1] that represents a very flexible and useful tool for handling modifications to schemas with associated documents. The E$\mathcal{X}$up system extends the X-Evolution system [3] by introducing support for XSUpdate statements and providing different approaches for making effective the updates in off-the-shelf native DBMSs.

In the remainder of the paper, Section II briefly presents XSUpate providing some example statements on a running example. The characteristics of the E$\mathcal{X}$up engine are summarized in Section III. Section IV presents the features of the system that will be demonstrated.

II. XSUpdate in a Nutshell

XSUpdate is an SQL-like language designed to express evolution statements over an XML Schema. Every schema modification operation is executed over a set of one or more components in a specific schema, named the *evolution objects*. An evolution object can be the root element, a type definition or the declaration of an attribute, element or grouping operator (sequence, choice or all).

An XSUpdate statement is composed of three parts: (i) the identification of the evolution objects in a schema, (ii) the specification of the modification operation to carry out on the evolution objects and (iii) the specification of the impact of the update on documents (document adaptation). In the following we detail these components and provide some complete examples.

A. Evolution object

Starting from the text representation of an XML Schema, an abstract graph representation can be easily obtained that points

```xml
<?xml version="1.0" encoding="UTF-8"?>
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema">
  <xs:element name="Log" type="LogType"/>
  <xs:complexType name="LogType">
    <xs:sequence maxOccurs="unbounded">
      <xs:choice>
        <xs:element name="DestIP" type="IPAddressType"/>
        <xs:element name="SourceIP" type="IPAddressType"/>
      </xs:choice>
      <xs:element name="Cookies" type="xs:string" minOccurs="0"/>
      <xs:element ref="Packet" maxOccurs="unbounded"/>
    </xs:sequence>
    <xs:attribute name="SchemaVersion" type="xs:string" use="required"/>
  </xs:complexType>
  <xs:simpleType name="IPAddressType">
    <xs:restriction base="xs:string">
      <xs:minLength value="7"/>
      <xs:maxLength value="15"/>
    </xs:restriction>
  </xs:simpleType>
  <xs:element name="Packet" type="PacketType"/>
  <xs:complexType name="PacketType">
    <xs:simpleContent>
      <xs:extension base="xs:string">
        <xs:attribute name="Date" type="xs:string" use="required"/>
        <xs:attribute name="Time" type="xs:string" use="required"/>
      </xs:extension>
    </xs:simpleContent>
  </xs:complexType>
</xs:schema>
```

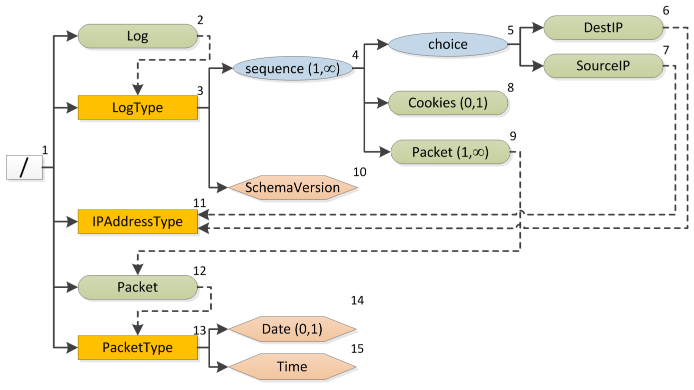


Fig. 1. XML Schema `HTML-Log.xsd` (left) and graph representation.

out the hierarchical organization of a schema plus the implicit relationships due to the presence of elements or attributes that need to be bound with their global types or referred global declarations. Fig. 1 shows a simple schema with its graph representation (solid arrows represent the hierarchical organization, while dashed ones the implicit relationships).

Relying on this representation, the schema components to be updated or the corresponding elements/attributes in documents associated with the schema are identified by means of navigational expressions specified in XSPath [2]. By means of this language concise and intuitive navigational expressions can be specified on a schema, being able, for instance, to retrieve the elements in a declaration regardless on how their type has been actually defined. XSPath relies on a compositional semantic similar to that of XPath but using the names specified in declarations and definitions as primary identification means. As in XPath, axes are employed to identify nodes in a specific relation, while node selectors and predicates can be used to identify only those nodes exhibiting specific properties.

B. Schema modification

The second part of an XSUpdate statement specifies the modification primitive to apply on the identified components of the schema (element/attribute declaration, simple/complex type definition). The set of primitives includes those for inserting/deleting/modifying the evolution object, those for moving the evolution object into another position in the schema, and those for migrating a local type/element into a global one or vice versa. Each primitive is associated with a set of applicability conditions that guarantee that the modified schema is still well-formed according to the W3C specification.

C. Document adaptation

Schema modifications can invalidate associated documents, therefore XSUpdate offers three different approaches to handle them: (i) documents can be left unmodified, (ii) an automatic approach can be applied to minimally change the documents in order to make them valid for the updated schema, or (iii) the entire adaptation process can be controlled by the user through XQU expressions. After the adaptation, the parts of the documents affected by the schema evolution are revalidated and those still invalid can be disassociated from the schema or the whole modification can be rolled back.

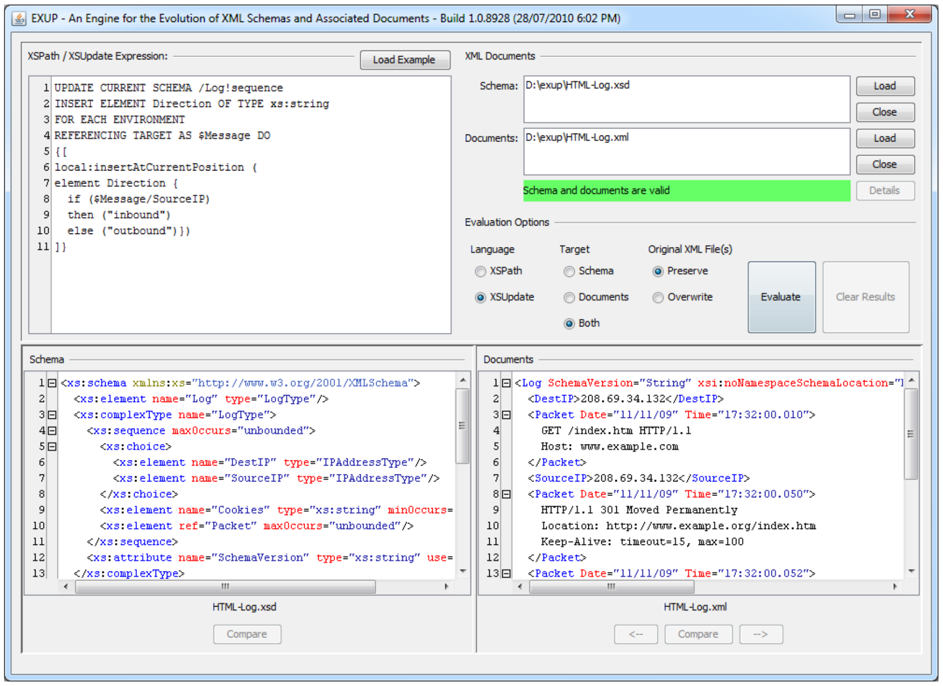
The first and simplest approach is to leave documents unaltered. To identify the documents no longer valid, they should go through a re-validation. The knowledge of the applied schema modification allows an incremental revalidation only on the parts of the documents affected by the modification.

The automatic adaptation approach follows two major guidelines: (i) make smallest modification to document nodes affected by the modifications to re-establish the document validity; (ii) insertions of new nodes, alterations or removal of existing data are realized only when strictly needed. This approach guarantees document validity after the adaptation.

A user-defined document adaptation can finally be specified, allowing the user to specify, through XQU expressions, ad hoc ways to convert old documents in documents valid for the new schema. Specifically, user-defined adaptation allows to specify an *iteration expression* and a *document expression*. The iteration and document expressions may present free variables that are automatically bound relying on the modification performed at schema level. The binding is realized through an *environment* that establishes the position within the document where the two expressions should be evaluated and the values to be assigned to the free variables.

D. Examples of XSUpdate statements

The following XSUpdate statements refer to the schema in Fig. 1. To modify the type definition `IPAddressType` replacing the `minLength` and `maxLength` restrictions with a `pattern` restriction and leave any document containing invalid IP addresses disassociated from the modified schema,



EXUP - An Engine for the Evolution of XML Schemas and Associated Documents - Build 1.0.8928 (28/07/2010 6:02 PM)

XSPath / XSUpdate Expression:

```
UPDATE CURRENT SCHEMA /Log!sequence
INSERT ELEMENT Direction OF TYPE xs:string
FOR EACH ENVIRONMENT
REFERENCING TARGET AS $Message DO
{[
local:insertAtCurrentPosition (
element Direction {
  if ($Message/SourceIP)
  then ("inbound")
  else ("outbound")})
]}
```

XML Documents

Schema: D:\exup\HTML-Log.xsd

Documents: D:\exup\HTML-Log.xml

Schema and documents are valid

Evaluation Options

Language: ( ) XSPath [✓ XSUpdate]

Target: ( ) Schema ( ) Documents [✓ Both]

Original XML File(s): [✓ Preserve] ( ) Overwrite

Schema

```
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema">
  <xs:element name="Log" type="LogType"/>
  <xs:complexType name="LogType">
    <xs:sequence maxOccurs="unbounded">
      <xs:choice>
        <xs:element name="DestIP" type="IPAddressType"/>
        <xs:element name="SourceIP" type="IPAddressType"/>
      </xs:choice>
      <xs:element name="Cookies" type="xs:string" minOccurs=
      <xs:element ref="Packet" maxOccurs="unbounded"/>
    </xs:sequence>
    <xs:attribute name="SchemaVersion" type="xs:string" use=
  </xs:complexType>
```

HTML-Log.xsd

Documents

```
<Log SchemaVersion="String" xsi:noNamespaceSchemaLocation="]
  <DestIP>208.69.34.132</DestIP>
  <Packet Date="11/11/09" Time="17:32:00.010">
    GET /index.htm HTTP/1.1
    Host: www.example.com
  </Packet>
  <SourceIP>208.69.34.132</SourceIP>
  <Packet Date="11/11/09" Time="17:32:00.050">
    HTTP/1.1 301 Moved Permanently
    Location: http://www.example.org/index.htm
    Keep-Alive: timeout=15, max=100
  </Packet>
  <Packet Date="11/11/09" Time="17:32:00.052">
```

HTML-Log.xml

Fig. 2. The GUI.

the following statement can be specified:

```
UPDATE SCHEMA ("HTTP-Log.xsd")/#IPAddressType
REPLACE RESTRICTIONS minLength, maxLength
WITH pattern =
[0-9]{1-3}\.[0-9]{1-3}\.[0-9]{1-3}\.[0-9]{1-3}
NO ADAPT REMOVE INVALID
```

Types, elements, attributes, operators or group of nodes can be inserted specifying a position in the schema and providing their definition (which may include XML Schema fragments). Suppose we wish to insert an element `Direction` of type `xs:string` as last child in the `Log` declaration, and to adapt the associated documents inserting - where needed - a `Direction` element, whose value is determined depending on the occurrence of the `SourceIP` element. The following statement can be issued:

```
UPDATE SCHEMA ("HTTP-Log.xsd")/Log!sequence
INSERT ELEMENT Direction OF TYPE xs:string
FOR EACH ENVIRONMENT
REFERENCING TARGET AS $Message DO
local:insertAtCurrentPosition
    (element Direction {
       if ($Message/SourceIP)
       then ("inbound")  else ("outbound")})
```

The statement is composed of two parts: the modification of the schema and the XQU expression to be evaluated in each environment bound by the `$Message` variable.

Types, cardinality specifications, and names can be changed as needed. The following statement requires to change the cardinality of the `Cookies` element from `0,1` to `1,1`. Relying on the automatic adaptation option, a new `Cookies` element is inserted in each message without one. The value of the inserted elements is the empty string (the default value for the `string` type).

```
UPDATE SCHEMA ("HTTP-Log.xsd")/Log/Cookies
CHANGE CARDINALITY TO 1,1
```

Any definition/declaration can be removed, provided that the resulting schema remains consistent. The following statement removes the type `IPAddressType` and any element or type depending on it.

```
UPDATE SCHEMA ("HTTP-Log.xsd")/#IPAddressType
REMOVE CASCADE
```

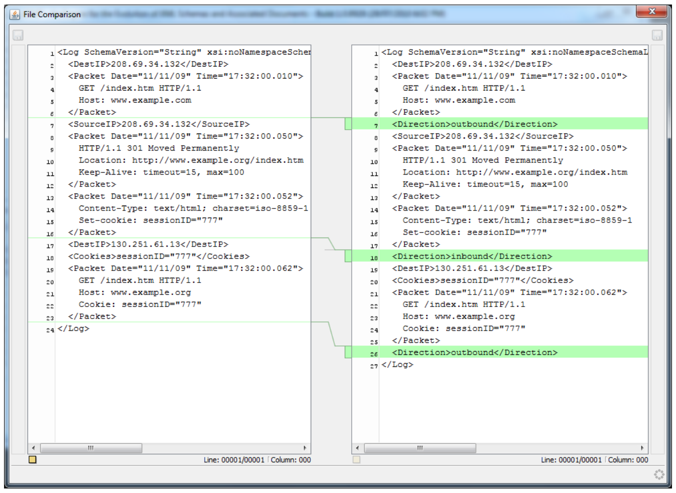
III. The E𝒳up Engine

E𝒳up is a Java application for the translation and evaluation of XSUpdate statements. Given an XSUpdate statement on a schema, it is translated in an XQU expression to be evaluated on the schema, and in an XQU expression to be evaluated on any of the documents associated with it. Key point of this last translation is the identification in the document of the environments where the iteration and document expressions must be evaluated. Moreover, XSPath expressions as well are translated into simple XPath expressions that can be evaluated on the schema or on associated documents.

E𝒳up offers two user interfaces, one applet-based for Web use and one for local use. E𝒳up implements the algorithms for the automatic adaptation, and for translating statements in corresponding XQU expressions; it can also be used to validate XML documents or parts of them. Other features include detailed syntactic and semantic error reporting and visual analysis of the modifications performed on a schema as well as on its associated documents. A comprehensive set of APIs is also available, offering all the features of the user interfaces and additional implementation-related options.

Document collections and schemas can be loaded from files or from an XML native or enabled DBMS. The translation of XSUpdate evolution statements on documents employs external Java functions to perform the environment identification process described above. An effort has been made to support all common Java XQuery Update libraries supporting external functions: Saxon EE, MXQuery, and Qizx/open. Without any optimization, the translation process of XSPath expressions in both XQuery and XPath, with respect to their length, requires linear time to be generated and produces expression of linear length. The translation of a schema modification, with respect to the XSUpdate statement length, has linear length and requires linear time. To translate and evaluate an XSUpdate statement against a schema usually require no more than a tenth of a second (with schemas of size 100KB). We also compared the overall performance of the translation plus evaluation of a document adaptation specified as an XSUpdate statement with the evaluation of a hand-written XQU expression achieving the same goal. Note that, especially when optional surrounding elements have to be considered, such an equivalent hand-written expression might be too complex for most XQuery users. Even if the translation on average is no more than 20% slower, we are working to enhance this result.

Fig. 2 reports the main EXup GUI. A user can specify the schema and the documents on which she wishes to work on. The schema is loaded in the left bottom text box, whereas the documents are loaded in the right bottom text box (when more than one document is loaded, it is possible to access a specific one by means of arrows). When the documents are loaded, they are also validated against the schema. In the left top text box, the user can specify the XSPath expression or the XSUpdate statement. Then, by means of different check boxes, the user can specify the kind of query, whether it should be evaluated only on the schema, on the documents, or on both, and, in case of an update, make effective the modifications on the storage. Fig. 3 shows a comparison between the old and new documents due to the update required through the XSUpdate statement in Fig. 2.



```
<Log SchemaVersion="String" xsi:noNamespaceScher
  <DestIP>208.69.34.132</DestIP>
  <Packet Date="11/11/09" Time="17:32:00.010">
    GET /index.htm HTTP/1.1
    Host: www.example.com
  </Packet>
  <SourceIP>208.69.34.132</SourceIP>
  <Packet Date="11/11/09" Time="17:32:00.050">
    HTTP/1.1 301 Moved Permanently
    Location: http://www.example.org/index.htm
    Keep-Alive: timeout=15, max=100
  </Packet>
  <Packet Date="11/11/09" Time="17:32:00.052">
    Content-Type: text/html; charset=iso-8859-1
    Set-cookie: sessionID="777"
  </Packet>
  <DestIP>130.251.61.13</DestIP>
  <Cookies>sessionID="777"</Cookies>
  <Packet Date="11/11/09" Time="17:32:00.062">
    GET /index.htm HTTP/1.1
    Host: www.example.org
    Cookie: sessionID="777"
  </Packet>
</Log>
```

```
<Log SchemaVersion="String" xsi:noNamespaceSchemaL
  <DestIP>208.69.34.132</DestIP>
  <Packet Date="11/11/09" Time="17:32:00.010">
    GET /index.htm HTTP/1.1
    Host: www.example.com
  </Packet>
  <Direction>outbound</Direction>
  <SourceIP>208.69.34.132</SourceIP>
  <Packet Date="11/11/09" Time="17:32:00.050">
    HTTP/1.1 301 Moved Permanently
    Location: http://www.example.org/index.htm
    Keep-Alive: timeout=15, max=100
  </Packet>
  <Packet Date="11/11/09" Time="17:32:00.052">
    Content-Type: text/html; charset=iso-8859-1
    Set-cookie: sessionID="777"
  </Packet>
  <Direction>inbound</Direction>
  <DestIP>130.251.61.13</DestIP>
  <Cookies>sessionID="777"</Cookies>
  <Packet Date="11/11/09" Time="17:32:00.062">
    GET /index.htm HTTP/1.1
    Host: www.example.org
    Cookie: sessionID="777"
  </Packet>
  <Direction>outbound</Direction>
</Log>
```

Fig. 3. Comparison between the old and new document.

IV. Demonstration

The demo shows how easy it is to specify XSUpdate statements using the EXup engine and to check the effects of modifications on the associated documents by means of the EXup GUIs. Specifically:

- A set of XSPath expressions will be evaluated. For each of them, the system will outline the identified evolution object in the schema and the corresponding elements/attributes in the associated documents. Moreover, the corresponding XPath expressions that can be evaluated on the schema and documents to obtain the same result will be shown.
- A set of XSUpdate statements will be evaluated to show the functionalities of the language in updating a schema and the associated documents. For each of them, we will graphically show the effects on the schema (by comparing the new and old versions of the schema), and on the associated documents (highlighting through colors - green and red, respectively - inserted and deleted nodes).
- We will show the corresponding XPath and XQU expressions that the system generates for being evaluated in the Qizx engine and the difference between them and those we have explicitly hand-written for the same purpose.
- We will show how the generated XQU expression can be locally evaluated and a corresponding PUL generated. This functionality can be useful when the updates on the current documents/schema should be propagated to a third party repository holding their master copy.

The EXup tool is, to the best of our knowledge, the only tool supporting XML Schema evolution and associated document adaptation. The associated XSUpdate language is highly flexible in supporting both automatic and user-defined arbitrary document adaptations. A language usability analysis demonstrated that, also for experienced XQuery users, denoting schema components, specifying their modifications and corresponding document adaptations in XSUpdate is much more intuitive and less error prone than using XPath/XQU on the XML documents representing XML Schemas. On the other side, experimental performance evaluation demonstrated that the overhead imposed by the translation from XSPath/XSUpdate to XPath/XQU is tolerable.

Interested readers can play with EXup through the following website `felix.disi.unige.it/exup` by loading the java applet contained in the demo section.

References

[1] F. Cavalieri. EXup: An Engine for the Evolution of XML Schemas and associated Documents. EDBT/ICDT Workshops 2010

[2] F. Cavalieri, G. Guerrini, and M. Mesiti. Navigational Path Expressions on XML Schemas. In *DEXA*, LNCS(5181), 718–726. 2008.

[3] G. Guerrini, and M. Mesiti. X-Evolution: A Comprehensive Approach for XML Schema Evolution. In *DEXA workshop*, 251-255. 2008.

[4] W3C, "XQuery Update Facility 1.0," `http://www.w3.org/TR/xquery-update-10/`, June 2009.