On Different Perspectives of XML Data Evolution*

Martin Nečaský
Dept. of Software Science and Engineering
Faculty of Electrical Engineering
Czech Technical University, Prague, Czech Rep.
E-mail: necasky@fel.cvut.cz

Irena Mlýnková
Dept. of Software Engineering
Faculty of Mathematics and Physics
Charles University, Prague, Czech Rep.
E-mail: mlynkova@ksi.mff.cuni.cz

Abstract

XML data evolution has recently gained much interest in both research and practice. However, most of the existing works deal with separate aspects of the problem such as evolution of XML schemas or evolution of conceptual schemas and view the problem only from the perspective of a single application. In this paper we show that XML data evolution has several different levels at which it can be performed and that are highly related. Secondly, we show that evolution is not the problem of a single application, but multiple applications having the same problem domain can influence each other as well. We describe the particular levels, how they can be modified and the respective propagation of the modifications to other levels and applications.

1 Introduction

Since XML [5] has become a de-facto standard for data representation and manipulation, there exists a huge amount of applications having their data represented in XML. Since most of the XML applications are usually dynamic, sooner or later the structure of the data needs to be changed. We usually speak about so-called *XML schema evolution*, i.e. a situation that a schema of the data is updated and we need to apply these updates on its existing instances.

Currently there exist several works focusing on this topic. However, most of them deal with separate aspects such as evolution of XML schemas or evolution of conceptual schemas. In addition, all of them view the problem only from the perspective of a single application. Hence, in this paper we study the problem from a more general perspective. We show that schema evolution has several different levels at which it can be performed and that are highly related. Secondly, we show that schema evolution is not the problem of a single application, but multiple applications having the same problem domain can influence each other as well. We describe the particular levels, how they can be modified and the respective propagation of the modifications to other levels and applications.

The paper is structured as follows: Section 2 overviews existing related works. Section 3 introduces the given problem using a motivating example. Section 4 describes the proposed evolution model and Section 5 the respective transformations. Finally, Section 6 provides conclusions and outlines future work.

2 Related Work

We can divide current approaches to XML schema evolution into three groups depending on the level where transformations can be specified by the designer.

Approaches in the first group consider transformations at so-called *logical*, i.e. XML schema level. They differ mainly in the selected XML schema language, i.e. DTD [12, 2] or XML Schema [13, 9]. The transformations are then propagated to the respective XML documents, i.e. to so-called *extensional* level, to ensure that XML documents are valid against the evolved XML schema. There also exists an opposite approach that enables to evolve XML documents and propagate the transformations to their XML schema [3].

Approaches in the second and third group consider transformations specified at an abstraction of the logical level. In particular, approaches in the second group consider a visualization of the XML schema [8], whereas approaches in the third group consider a UML class diagram that models the XML schema [7]. From our point of view, we comprehend both the cases as so-called *platform-specific* model, since it directly models

*This paper was partially supported by the MSMT grant No. MSM 6840770014. This research has also been partially supported by GAČR grants No. 201/09/0990 and 201/09/P364.

the components of the XML schema but not the data abstracted from its representation in XML. Transformations made at platform-specific level are propagated to XML schema and then to XML documents. The main advantage is that it is easier for a designer to specify transformations at the abstraction level, because (s)he can concentrate more on the transformation itself rather than XML schema technical details.

Another open problem related to schema evolution is adaptation of the respective operations, in particular XML queries. We speak about so-called *operational* level. There seems to exist only a single paper [10] dealing with this topic which states several recommendations how to write queries that do not need to be adapted for an evolving schema.

3 Motivating Example

Let us consider a company that sells products to customers. The information system of the company is composed of several applications utilized by different users for different purposes. Each application applies different XML formats that represent different user views of the data. On the other hand, since the applications work with the same data, e.g. customers, products, etc., the XML formats represent the same *problem domain*.

For example we consider two particular XML formats, each described by an XML schema in the XML Schema language (XSD) depicted in Figure 1. The first XSD `PurchaseRequest.xsd` describes an XML format used by customers to send purchase requests to the company. The second XSD `SalesReport.xsd` describes a format representing reports on sales of products in regions. Both XML schemas somehow represent customers. While `PurchaseRequest.xsd` represents only their registration numbers, `SalesReport.xsd` represents registration numbers as well as names.

In our first scenario we need the customers to specify their names in purchase requests. Since `PurchaseRequest.xsd` does not consider names we have to modify it. We decide to add subelement `name` to element `purchase-request`. This transformation does not change our interpretation of the domain since we have already considered names of customers. It does not therefore influence `SalesReport.xsd`.

Secondly, sales managers want to structure names of customers in sales reports to first and family names. `SalesReport.xsd` has only XML element `name` in XML element `customer`. Therefore, we replace `name` with `first-name` and `family-name`. This transformation, however, changes our interpretation of the problem domain since we have considered a name as a single unstructured value. This can influence other XSDs as well. In particular, we need to structure XML element `name` in `PurchaseRequest.xsd` in a similar way.

```
<xs:schema xmlns:xs=“.../XMLSchema″>
 <xs:element name=″purchase-request″
             type=″Purchase″/>
 <xs:complexType name=″Purchase″>
  <xs:sequence>
   <xs:choice>
    <xs:element name=″messenger″
                type=″Messenger″
                minOccurs=″0"/>
    <xs:element name=″van″
                type=″Van″
                minOccurs=″0"/>
   </xs:choice>
   <xs:element name=″item″
               type=″Item″
               maxOccurs=″unbounded″/>
  </xs:sequence>
  <xs:attribute name=″issue-date″
                type=″xs:string″/>
  <xs:attribute name=″registration″
                type=″xs:string″/>
 </xs:complexType>
 <xs:complexType name=″Messenger″>
  <xs:attribute name=″code″
                type=″xs:string″/>
 </xs:compleType>
 <xs:complexType name=″Van″>
  <xs:attribute name=″plate-no″
                type=″xs:string″/>
 </xs:compleType>
 <xs:complexType name=″Item″>
  <xs:sequence>
   <xs:element name=″amount″
               type=″xs:string″/>
   <xs:element name=″price″
               type=″xs:string″/>
  </xs:sequence>
  <xs:attribute name=″code″
                type=″xs:string″/>
 </xs:compleType>
</xs:schema>
```

```
<xs:schema xmlns:xs=″.../XMLSchema″>
 <xs:element name=″sales-report″
             type=″Product″/>
 <xs:complexType name=″Product″>
  <xs:sequence>
   <xs:element name=″name″
               type=″xs:string″/>
   <xs:element name=″region″
               type=″Region″
               minOccurs=″0"
               maxOccurs=″unbounded″/>
  </xs:sequence>
  <xs:attribute name=″code″
                type=″xs:string″/>
 </xs:complexType>
 <xs:complexType name=″Region″>
  <xs:sequence>
   <xs:element name=″customer″
               type=″Customer″
               minOccurs=″0"
               maxOccurs=″unbounded″/>
  </xs:sequence>
  <xs:attribute name=″code″
                type=″xs:string″/>
 </xs:complexType>
 <xs:complexType name=″Customer″>
  <xs:sequence>
   <xs:element name=″name″
               type=″xs:string″/>
   <xs:element name=″email″
               type=″xs:string″/>
  </xs:sequence>
  <xs:attribute name=″registration″
                type=″xs:string″/>
 </xs:complexType>
</xs:schema>
```

Figure 1. XSDs

As the examples demonstrate, a transformation of an XML schema can influence other XML schemas in the system as well. However, managing the consistency at the XML schema level is an error-prone and time-consuming task since it must be handled manually. Therefore, a more sophisticated method for XML schema evolution would be useful in practice.

4 Five-Level Schema Evolution

While the current approaches consider evolution on two, resp. three levels, in this paper we consider five levels. The introduced levels are depicted in Figure 2.

The *extensional level* contains for each XML format a set of XML documents conforming to this format. The *logical level* contains an XML schema that describes structure of the XML format. The *operational level* contains queries related to the XML format. If we consider only these levels, it is hard to determine whether a transformation of an XML format influences other XML formats since it must be determined manually. Therefore, we add two other levels, i.e. platform-specific and platform-independent level, that provide their interrelation. They are adopted from the Model-driven Architecture (MDA) terminology which consid-

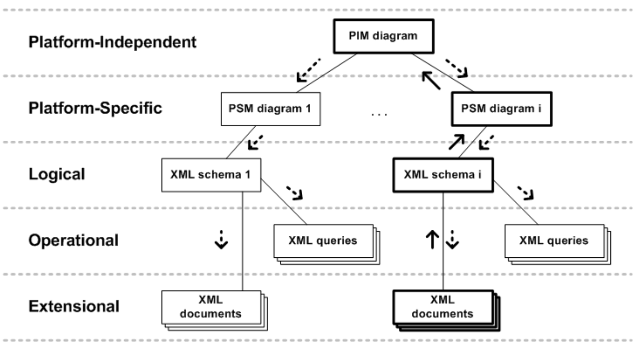

Figure 2. Five-level evolution architecture

ers modeling data at different levels of abstraction. The *platform-independent level* contains a conceptual diagram of the problem domain. It provides a description of the problem domain abstracted from the logical level. The *platform-specific level* interrelates the platform-independent and logical level. It provides a mapping of each XML schema to the conceptual diagram. This mapping enables automatic transformation propagation since a transformation of an XML schema can be propagated to the conceptual diagram and from here to other XML schemas.

Figure 2 depicts in bold rectangles that a transformation can occur at any level L except the operational level. From L, the transformation is firstly propagated to the upper levels. This propagation is therefore called *upward propagation* and is depicted in Figure 2 by solid arrows. After the upward propagation, the transformation is propagated back to the lower levels as depicted in Figure 2 by dashed arrows. This propagation is therefore called *downward propagation*.

The upward propagation means that a transformation specified at L implies a corresponding transformation at the upper level $L-1$. This propagation does not occur in every case. There are transformations whose impact depends on a designer or that have no impact at all. The upward propagation can end up at any level from where the downward propagation continues. If the upward propagation ends up at the platform-specific or lower level, the downward propagation continues only in the scope of the corresponding XML format. If the propagation ends up at the platform-independent level, our interpretation of the problem domain has changed and a downward propagation to other XML formats can be therefore required.

Platform-Independent Level The platform-independent level contains a conceptual diagram of the problem domain. It describes the domain independently of the considered XML formats. To design such a diagram we use a conceptual modeling language which is called *Platform-Independent Model (PIM)* in the MDA terminology. The diagram is then called *PIM diagram.* As PIM, we consider the well-known UML class model. We consider only basic modeling constructs, i.e. classes for modeling concepts and binary associations for modeling relationships between the concepts.

Example 1 *Figure 3 shows a sample PIM diagram modeling the domain of the company introduced in Section 3 designed in a tool called XCase [1] we have developed for conceptual modeling of XML schemas. There is a class Customer modeling customers. It has attributes registration, name, email and phone modeling relevant customer characteristics. A sample association is the one connecting Customer and Purchase. It models that customers make purchases.*

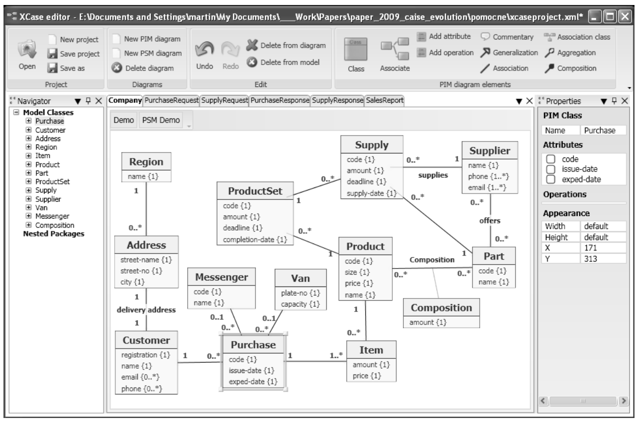

Figure 3. PIM diagram

Platform-Specific Level The platform-specific level contains for each XML format a diagram that models the XML format in terms of the PIM diagram. It serves as a mapping between the corresponding XML schema and the PIM diagram. For this, we use a modeling language which is called *Platform-Specific Model (PSM)* in the MDA terminology. The resulting diagram is called *PSM diagram.* As PSM, we consider the UML class model extended with some constructs covering XML-specific features. In this paper, we introduce only some of the constructs. For their full description, we refer to [11].

A PSM diagram contains classes from the PIM diagram and organizes them into the hierarchical structure of the modeled XML format by associations. There is a formal background that maps associations in PSM diagrams to associations in a PIM diagram. However, we omit it in this paper. A label displayed above a class is called *element label* and specifies a name of an XML element that represents instances of the class in the XML format. Attributes of a class model XML attributes. They can be separated to so-called *attribute container* displayed by a box beneath the class. In that case they model XML elements. We can also model variants in the content of a PSM class by so-called *content choice.* It is displayed by a circle with an inner | and models that only one of its components can be instantiated for each its instance.

Example 2 *Figure 4 shows two sample PSM diagrams modeling the XML formats for purchase requests and sales reports from Section 3. Both model the respective XML format in terms of the PIM diagram depicted in Figure 3. The PSM diagrams were designed in the XCase tool [1].*

To explain the PSM constructs, consider the diagram on the left. It models that purchases are represented in the corresponding XML format as roots since Purchase is a root class. Because the root PSM class Purchase has an element label purchase-request, a purchase is represented as an XML element `purchase-request`*. The associations going from Purchase model that a purchase has a customer, delivery information and list of items as children whose representation is modeled by the children of Purchase. The diagram utilizes an attribute container to specify that attributes amount and price of Item model XML elements. It also utilizes a content choice to specify that each purchase contains a messenger or van, that delivers the purchase, but not both.*

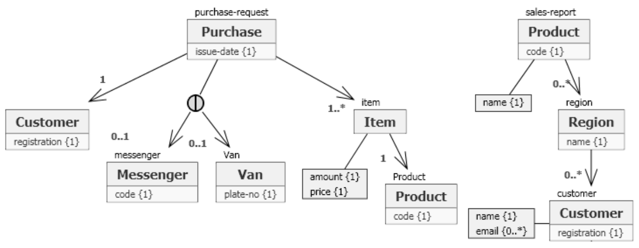

Figure 4. PSM diagrams

Logical, Extensional and Operational Level The logical level contains for each considered XML format its XML schema. In particular we consider the XML Schema language [14, 4]. The extensional level contains for each XML format a set of XML documents. And, finally, the operational level contains for each XML format queries that are evaluated over the XML documents of that format. There exist several XML query languages, however they all exploit XPath [6] to navigate in the structure of XML documents. Therefore, especially XPath expressions must be considered during evolution since a change in the structure of an XML format can have an impact on them.

5 Transformations and Propagation

Having the described levels, we can now specify the set of transformations a user may invoke at each of them. Similarly to existing works [9] we can distinguish *atomic* transformations and *high-level* transformations (i.e. sequences of atomic transformations). We further adopt more specific types of transformations [10]:

- Structural:
  - *Adding* – adds a new item
  - *Removal* – removes a new item
- Sedentary:
  - *Extension* – adds a new item that does not change structure
  - *Renaming* – renames an item
  - *Renumbering* – changes the cardinality of an item
  - *Retyping* – changes the data type of an item
  - *Resetting* – changes the value of an item
  - *Mapping* – maps an item to an item from another level
  - *Unmapping* – removes a mapping between levels
- Migratory:
  - *Moving* – moves an item
  - *Reordering* – changes the order of items
  - *Transformation* – transforms an item to an item of a different type

Note that not all types of the transformations exist at all five levels. For instance, retyping does not occur at the platform-independent level, since we restrict ourselves only to classes, attributes and associations. Since the propagation of transformations can be either upwards or downwards, the transformations at particular levels need to be propagated to all neighboring levels. In general the propagation needs to be done if the XML data become invalid. However, in all the other cases it remains user's decision if the respective propagations should be done.

Let us consider the transformation propagation for the transformations introduced in Section 3. The first transformation was adding of new subelement `name` to `purchase-request`. We initiate the transformation at the extensional level by modifying an XML document. It must be then propagated to the logical level and the corresponding XML schema must be extended with the element declaration. Next, it must be propagated to the corresponding PSM diagram where we add a new attribute *name* to the class *Customer*. Finally, the transformation is propagated to the PIM diagram. Since customer names are already modeled by attribute *name* of *Customer* we just map the *name* PSM attribute to the existing *name* PIM attribute and no transformation is needed. Therefore, our interpretation of the domain has not been changed by the initial requirement and no downward propagation is required.

The second transformation was replacement of XML element `name` with new elements `first-name` and `family-name`. We initiate this transformation in `SalesReport.xsd`, i.e. at the logical level. In the corresponding PSM diagram, it means to replace the attribute *name* of *Customer* with new attributes *first-name* and *family-name*. Since these attributes have no equivalents in the PIM diagram, our interpretation of the domain has changed and in the PIM diagram the attribute *name* of *Customer* must be replaced with new attributes *first-name* and *family-name*. Consequently, the downward propagation must follow. It means to propagate the transformation to the XML documents with sales reports and, moreover, to the other XML formats, i.e. purchase requests. Since there can be queries for each XML format querying names of customers, they can be influenced as well.

6 Conclusion

The aim of this paper was to show that the problem of XML schema evolution has been highly marginalized so far. We have showed that the schema evolution problem must be viewed from the perspective of multiple applications and we have defined five levels of XML schema evolution that cover all the existing works. Next, we have described use cases related to the problem that cannot occur unless we consider all the evolution levels. Currently, we are dealing with a throughout implementation of the proposed system. For this purpose we extend system *XCase* [1] which enables to design conceptual diagrams and map them to respective XML schemas.

Apparently, there also exist several open issues: Similarly to the existing works we need to make the adaptation at all levels efficient, i.e. we need to be able to find the least expensive sequence of transformations. Secondly, to perform the transformations we want to output a set of XSLT scripts that can be applied on the respective XML data. Naturally, this approach requires the existence of an XML representation of the non-XML levels. And, having such a robust system, it is quite natural that there are multiple users to work with it, i.e. multi-user access and transactions need to be incorporated as well.

References

[1] *XCase – A Tool for XML Data Modeling.* 2008. `http://www.ksi.mff.cuni.cz/~necasky/xcase/`.

[2] L. Al-Jadir and F. El-Moukaddem. Once Upon a Time a DTD Evolved into Another DTD... In *Object-Oriented Information Systems*, pages 3–17, Berlin, Heidelberg, 2003. Springer.

[3] B. Bouchou et al. Schema Evolution for XML: A Consistency-Preserving Approach. In *Mathematical Foundations of Computer Science*, pages 876–888, Prague, Czech Republic, 2004. Springer-Verlag.

[4] P. V. Biron and A. Malhotra. *XML Schema Part 2: Datatypes (Second Edition).* W3C, October 2004.

[5] T. Bray, J. Paoli, C. M. Sperberg-McQueen, E. Maler, and F. Yergeau. *Extensible Markup Language (XML) 1.0 (Fourth Edition).* W3C, September 2006.

[6] J. Clark and S. DeRose. *XML Path Language (XPath) Version 1.0.* W3C, November 1999.

[7] E. Dominguez, J. Lloret, A. L. Rubio, and M. A. Zapata. Evolving XML Schemas and Documents Using UML Class Diagrams. In *DEXA '05*, pages 343–352, Berlin, Heidelberg, 2005. Springer.

[8] M. Klettke. Conceptual XML Schema Evolution - the CoDEX Approach for Design and Redesign. In *BTW Workshops*, pages 53–63. Aachen, 2007.

[9] M. Mesiti, R. Celle, M. A. Sorrenti, and G. Guerrini. X-Evolution: A System for XML Schema Evolution and Document Adaptation. In *EDBT '06*, pages 1143–1146, Berlin, Heidelberg, 2006. Springer.

[10] M. M. Moro, S. Malaika, and L. Lim. Preserving XML Queries During Schema Evolution. In *WWW '07*, pages 1341–1342, New York, NY, USA, 2007. ACM.

[11] M. Necasky. *Conceptual Modeling for XML.* PhD thesis, Charles University, 2008. `http://www.ksi.mff.cuni.cz/~necasky/thesis.pdf`.

[12] H. Su, D. K. Kramer, and E. A. Rundensteiner. XEM: XML Evolution Management. Technical Report WPI-CS-TR-02-09, Worcester Polytechnnic Institute, Worcester, Massachusetts, 2002.

[13] M. Tan and A. Goh. Keeping Pace with Evolving XML-Based Specifications. In *EDBT '04 Workshops*, pages 280–288, Berlin, Heidelberg, 2005. Springer.

[14] H. S. Thompson, D. Beech, M. Maloney, and N. Mendelsohn. *XML Schema Part 1: Structures (Second Edition).* W3C, October 2004.