Five-Level Multi-Application Schema Evolution[⋆]

Martin Nečaský, Irena Mlýnková

Charles University, Faculty of Mathematics and Physics,
Department of Software Engineering
Malostranské nám. 25, 118 00 Prague 1, Czech Republic
{necasky,mlynkova}@ksi.mff.cuni.cz

Abstract. Schema 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. In addition, all of them view the problem only from the perspective of a single application.
In this paper 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. In particular we deal with five levels – extensional, operational, logical, platform-specific and platform-independent. We describe the particular levels, how they can be modified and the respective propagation of the modifications to other levels and applications. We also show which of the situations have already been discussed and solved in the existing works as well as which of them still remain open.

1 Introduction

Since XML [6] has become a de-facto standard for data representation and manipulation, there exists a huge amount of applications having their data represented in XML. Naturally, each of the applications can work only with correct data. We distinguish two levels of such correctness – well formedness and validity. An XML document is *well-formed* if it meets the well-formedness constraints stated in XML specification [6] such as, e.g., that the tags ensure the correct tree structure. An XML document is *valid* if it is associated with a legal XML schema (i.e. a schema that conforms to the specification of the selected language, such as, e.g., DTD [6], XML Schema [18, 3], etc.), and complies with the constraints expressed in it.

On the other hand, since most of the XML applications are usually dynamic, sooner or later the structure of the data needs to be changed. At the same time, we still need to be able to work with the old as well as new data without any loss. In relation to this topic, we usually speak about so-called *schema evolution*, i.e. a situation that a schema of the data is updated and we need to apply

⋆ This work was supported in part by the Czech Science Foundation (GAČR), grants number 201/09/P364 and 201/09/0990.

these updates on its existing instances to revalidate them. In case of XML data, having a set of *XML schema transformations*, we can transform an XML schema S^x to an XML schema S^y. A natural requirement for such transformation is *consistency*, i.e. transforming a legal schema S^x to a legal XML schema S^y. The problem of XML schema evolution deals with applying a set of respective *XML data transformations* on XML documents valid against S^x so that they become valid against S^y.

Currently there exist several works dealing with this topic. However, most of the existing works deal with separate aspects of the problem 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. In particular we deal with five levels – extensional, operational, logical, platform-specific and platform-independent. We describe the particular levels, how they can be modified and the respective propagation of the modifications to other levels and applications. We also show which of the situations have already been discussed and solved in the existing works as well as which of them still remain open.

The paper is structured as follows: Section 2 introduces the given problem using a motivating example. Section 3 overviews existing related works. Section 4 describes the proposed evolution model in detail and Section 5 provides an overview of respective schema transformations. Section 6 overviews some real examples. And, finally, Section 7 provides conclusions and outlines possible future work.

2 Motivation

To depict the issue we are focusing on, we first provide a real-world example. We consider a company that produces products and sells them to customers. An information system of the company is composed of various applications utilized by different users for different purposes. Each application applies different XML formats that represent different user views on 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. If a change in one XML format results into a change in our interpretation of the domain, it can require to make corresponding changes in other XML formats as well. We demonstrate some examples in the rest of this section.

We consider two particular XML formats each described by an XML schema expressed in the XML Schema language. A first XML schema `PurchaseRequest.xsd` describes an XML format used by customers to send purchase requests to the company. A second XML schema `SalesReport.xsd` describes a format representing reports on sales of products in regions. The XML schemas are depicted in Figure 1.

```
<xs:schema xmlns:xs=”w3.org/2001/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=”w3.org/2001/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>
```

Fig. 1. XML schemas for purchase requests and sales reports

In our interpretation of the domain, there are customers each having a registration number and name. Both XML schemas somehow represent customers. While `PurchaseRequest.xsd` represents only their registration numbers, `SalesReport.xsd` represents registration numbers as well as names.

We need customers to specify their names in purchase requests. Since `PurchaseRequest.xsd` does not consider names we have to modify it. We decide to add XML element `name` to root XML 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`.

Further, 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 one unstructured value. This can therefore influence other XML schemas as well. In a concrete, we need to structure XML element `name` in `PurchaseRequest.xsd` in a similar way.

As the examples demonstrate, a transformation of an XML schema can influence other XML schemas in the system as well. This is because we need to keep the XML schemas consistent with our interpretation of the problem domain. 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.

3 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. These groups are depicted in Figure 2 in columns (a), (b) and (c) respectively.

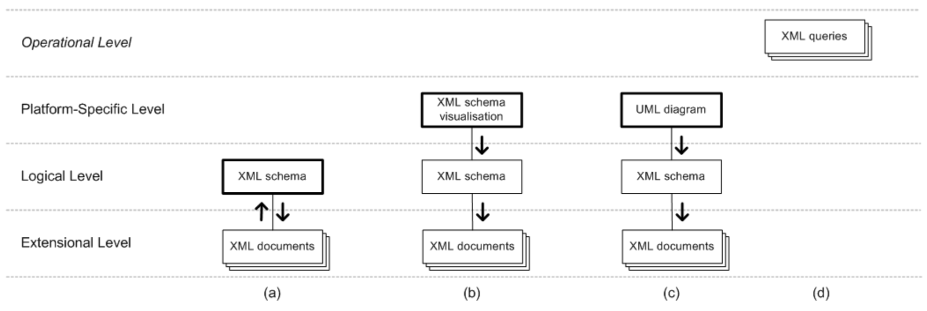

Fig. 2. Current XML Evolution Approaches

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 [16, 2, 9] or XML Schema [17, 12]. In general, the transformations can be variously classified. For instance, paper [17] proposes three classes: *Migratory* transformations deal with movements of elements/attributes to other parts of the schema or transformation of elements to attributes and vice versa. *Structural* transformations involve adding/removal of elements/attributes. Finally, *sedentary* transformations involve renaming of elements/attributes and modifications of simple data types. In all the proposed systems the transformations are then automatically propagated to the respective XML documents, i.e. to so-called *extensional* level, to ensure that the 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 [5].

Approaches in the second and third group are similar to the first one but they do not consider transformations specified at the logical level but its abstraction. In particular, approaches in the second group consider a visualization of the XML schema [11], whereas approaches in the third group consider a UML class diagram that models the XML schema [10]. From our point of view, we comprehend both the cases as so-called *platform-specific* model, since it directly models the components of the XML schema but not the data abstracted from its representation in XML. Transformations made at platform-specific level are then propagated to corresponding transformations in the XML schema and from here to XML documents. The main advantage is that it is easier for the designer to specify transformations at the abstraction level than at the XML schema level because (s)he can concentrate more on the transformation itself rather than technical XML schema details.

Another open problem related to schema evolution is adaptation the respective operations, in particular XML queries. We speak about so-called *operational* level (see Figure 2 by column (d)). Unfortunately, there seems to exist only a single paper [14] dealing with this topic. The authors provide several use cases when the queries need to be corrected with regard to the modification of the data and, finally, state several recommendations how to write queries that do not need to be adapted for evolving schema.

In general, the problem of current approaches is that they do not consider several important issues. Firstly, they consider evolution of a single XML format. However, as we demonstrated in Section 2, there are usually multiple XML formats providing different views of the domain. Hence, a transformation of an XML format can influence other XML formats as well. Secondly, queries over XML documents are ignored. Since queries depend on the structure of the XML documents, transformation of an XML schema can influence them and they should therefore be considered as a part of the evolution system as well.

4 Five Level Schema Evolution

In this paper, we introduce a new approach to XML evolution. While the current approaches consider evolution on two, resp. three levels as depicted in Figure 2, we consider five levels. The introduced levels are depicted in Figure 3.

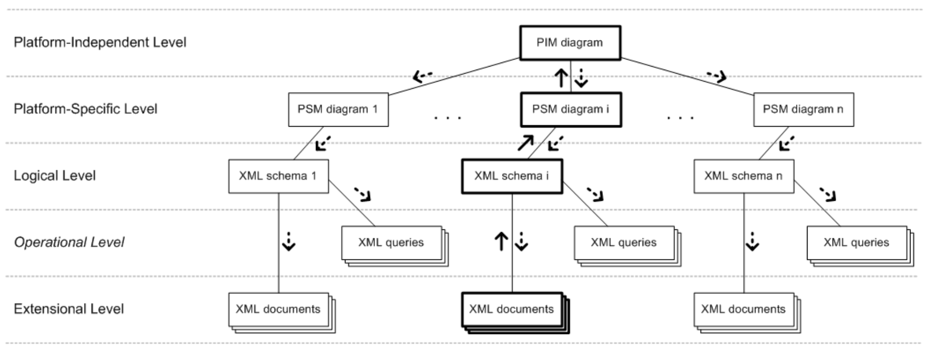

Fig. 3. Five level XML evolution architecture

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. The problem is that we have no layer that interrelates the XML formats. Therefore, we add two other levels, i.e. platform–specific and platform–independent level, that provide such interrelation.

We adopted the terms *platform–independent* and *platform–specific* from the Model–Driven Architecture (MDA) terminology which considers modeling data at different levels of abstraction. Even though MDA considers more levels, we adopt only the two mentioned ones. 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 diagram 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 3 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 3 by solid arrows. After the upward propagation, the transformation is propagated back to the lower levels as depicted in Figure 3 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. There are also transformations 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.

We describe possible transformations and their propagation in detail in Section 5. Before this, we describe the separate levels in the rest of this section.

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 4 shows a sample PIM diagram modeling the domain of the company introduced in Section 2. We designed the diagram in a tool called XCase [1] that we have developed for conceptual modeling of XML schemas. For example, 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.

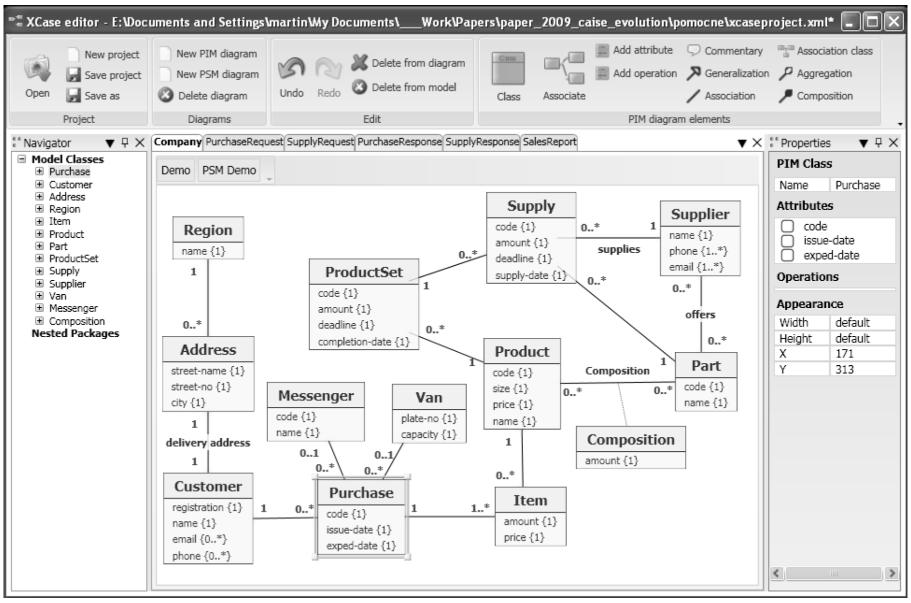

Fig. 4. Company PIM diagram

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. In other words, 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 [15].

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 instanciated for each its instance.

Example 2. Figure 5 shows two sample PSM diagrams modeling the XML formats for purchase requests and sales reports from Section 2. Both model the respective XML format in terms of the PIM diagram depicted in Figure 4. 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.

The PSM diagram provides a mapping of `PurchaseRequest.xsd` to the PIM diagram. Each PSM class maps a complex type in the XML schema to a corresponding PIM class. The same is for associations and attributes.

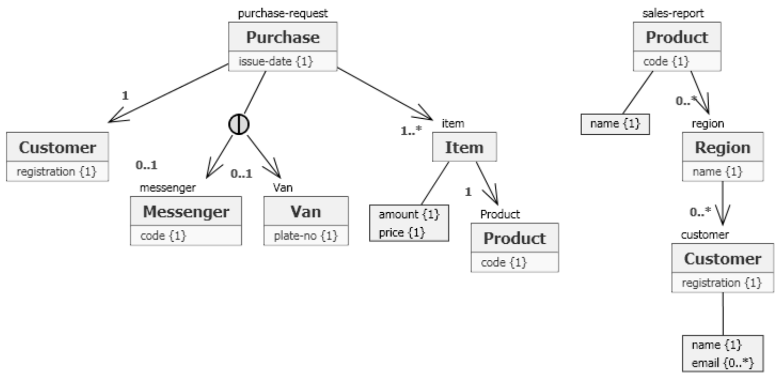

Fig. 5. Purchase and Sales Report PSM diagrams

Logical Level The logical level contains for each considered XML format its XML schema. An XML schema describes a syntactical structure, i.e. what XML elements and attributes can be used in respective XML documents. Even there are various languages for expressing XML schemas, we consider the XML Schema language [18, 3] in this paper. Self-descriptive examples of XSDs are depicted in Figure 1 in Section 2 .

XML Schema provides several constructs. We consider only the basic ones in this paper. Namely, simple data types (`simpleType`), complex data types (`complexType`), elements (`element`) and attributes (`attribute`). Since we are restricted in space, we do not describe the constructions in a more detail and refer to [18] for their explanation.

Extensional Level The extensional level contains for each considered XML format a set of XML documents. XML documents are composed of XML elements that can contain text values and/or nested XML elements. An XML element can also have XML attributes. Figure 6 shows two self-descriptive XML documents valid against the XML schemas depicted in Figure 1.

```
<purchase-request issue-date="30/11/2009"
                  registration="C828111">
 <messenger code="M190" />
 <item code="P2345">
  <amount>4</amount>
  <price>92</price>
 </item>
 <item code="P9982">
  <amount>1</amount>
  <price>10299</price>
 </item>
</purchase-request>
```

```
<sales-report code="P2345">
 <name>Product 2345</name>
 <region name="cz">
  <customer registration="C828111">
   <name>Martin Necasky</name>
   <email>martin@somewhere.cz</email>
  </customer>
  <customer registration="C802112">
   <name>Irena Mlynkova</name>
   <email>irena@somewhere.cz</email>
  </customer>
 </region>
 <!-- other regions -->
</sales-report>
```

Fig. 6. Sample XML documents conforming to XML formats for purchase requests and sales reports

Operational Level The operational level contains for each XML format queries that are evaluated on the XML documents of that format. There are several XML query languages such as, e.g. XSLT [7] or XQuery [4]. They all apply XPath [8] as a common factor to navigate in the structure of XML documents. Therefore, XPath expressions must be considered during evolution since a change in the structure of an XML format can have an impact on them. We do not describe XPath in detail here since we are restricted in space. A sample XPath expression is a path `/sales-report//customer/name` that targets names of customers in XML documents with sales reports.

5 Transformations and their Propagation

Having the above described levels, we can now specify the set of transformations a user may make at each of them. Similarly to the existing works [12] we can distinguish atomic transformations and high-level transformations (i.e. sequences of atomic transformations) as well as sedentary, structural and migratory ones. However, since this classification is too general, we further adopt general types of transformations similar to [14]:

- 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 a set 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 we are dealing with. For instance, retyping does not occur at the platform–indepdendent level, since we restrict ourselves only to classes, attributes and associations.

As we have described, propagation of transformations can be either upwards or downwards. Hence, the transformations at particular levels need to be propagated to all neighboring levels. With our general view of the data, we do not need to propagate the modifications to all the other levels if not necessary. 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. In the following text we describe only the necessary propagations, however also user-required propagations are discussed in important cases. For simplicity, we will deal only with atomic operations and thus omit all migratory ones that are high-level. For the same reason we omit trivial transformations, such as, e.g., renaming. (The full description can be found in [13].)

5.1 Platform–Independent Level

As we can see in Table 1, most of the operations at the platform–independent level are of the type of adding or removal that can be applied on all PIM items. Furthermore we can apply renumbering on the cardinalities. The operations can be propagated only to the platform–specific level within the downwards propagation.

Class	**Operations**	**PSM Propagation**
Adding	add class, attribute or association	
Removal	remove class, attribute or association	delete mapping
Renumbering	change class cardinality	

Table 1. PIM transformations and their propagation

Adding a new PIM item does not need to automatically trigger propagation to the platform–specific level. However, if such propagation is required, a user must state the required mapping to the respective PSM items since there usually exist multiple options. On the other hand, deleting of a PIM item invokes deleting of the respective mapping. However, a user may require also optional deleting of the respective item as well. Similarly, changing cardinality may invoke changing cardinality of the respective PSM item depending on user's requirements.

5.2 Platform–Specific Level

As stated in Table 2 at the platform–specific level there are several sedentary transformations, in particular changing cardinalities or data types and adding/deleting mapping to a PIM item. The rest of the transformations are structural since they create or delete respective PSM items.

Class	**Operations**	**Logical Propagation**	**PIM Propagation**
Mapping	map a PSM item to a PIM item		create a PIM item (O)
Unmapping	delete mapping		
Adding	add class, attribute, association, content choice or content container	add simple type, complex type, element, attribute, choice operator or sequence operator	
Removal	remove class, attribute, association, content choice or content container	remove simple type, complex type, element, attribute, choice operator or sequence operator	
Renumbering	change class cardinality	change element cardinality	
Retyping	change data type	change simple type	

Table 2. PSM transformations and their propagation

Except for the mapping transformations, all the others need to be propagated to the logical level. (Note that for the sake of simplicity we do not deal with XML schema specific aspects such as possibilities of creating a global or local element or equivalent content model. The detailed description of propagation of operations with particular schema items can be found in [13].) On the other hand, except for mapping a PSM item to a PIM item which requires creating of the respective PIM item if it does not exist yet, no other transformations need to be propagated from the platform–specific to platform–independent level if not otherwise specified by a user.

5.3 Logical Level

Table 3 provides an overview of logical-level transformations. Similarly to the previous case, for the sake of simplicity we omit all the possible specifications of an equivalent XML schema construct. To ensure consistency of the transformations we also state the following invariant that needs to hold during the whole evolution process: A globally defined item can be deleted only if all the respective references have already been deleted.

As we can see, most of the transformations belong to structural class since they add or delete XML schema items; the only sedentary transformations are changing a simple type or a cardinality and adding simple and complex types.

Class	**Operations**	**Extensional Propagation**	**PSM Propagation**
Adding	add element (+ simple or complex type), attribute (+ simple type), choice operator or sequence operator	add element or attribute (O)	add a class, attribute, association, content choice or content container (O)
Extension	add simple type or complex type		
Removal	remove simple type, complex type, element, attribute, choice operator or sequence operator	remove element or attribute (O)	delete class, attribute or content choice (O)
Renumbering	change element cardinality	add or remove element (O)	change class cardinality
Retyping	change simple type	change data value (O)	change data type

Table 3. Logical transformations and their propagation

As for the propagation to extensional level, all the transformations need to be propagated only in case the validity of XML documents is violated, i.e. all the propagations are optional (O). For instance, adding an XML schema element may cause adding an XML element only in case the XML schema element is compulsory. On the other hand, in case of logical-to-PSM propagation, we always need to propagate changes in data types and cardinalities. However, if we add or remove an XML schema item, the respective propagation depends on its type and can cause adding/removing of respective item(s) or nothing. For instance, if we create a complex type, the operation has no influence at platform–specific level, however if we create an element with a complex type, we need to create the respective PSM class. And, finally, note that creating a simple or complex type that is not associated with respective element or attribute has no influence in both directions of propagation.

5.4 Extensional Level

As we can see in Table 4 transformations over XML documents involve structural transformations of adding/removing an element/attribute and a sedentary transformation of changing a data value.

Depending on the XML schema, propagation of all the transformations is optional an depends on violation of validity. In addition, some of the transformations have several options how they can be propagated. For instance, creating

Class	**Operations**	**Logical Propagation**
Adding	add element or attribute	create element or attribute, change cardinality (O)
Removal	remove element or attribute	remove element or attribute, change cardinality (O)
Value Change	change data value of element or attribute	change simple data type (O)

Table 4. Extensional transformations and their propagation

an element may cause creating an element in the XML schema, changing cardinality or even nothing.

5.5 Operational Level

Last but not least, let us briefly discuss the operational level. In particular, considering only the XML data, we may restrict the transformation set to XML queries. The question is how can be this level influenced by the other levels and, conversely, whether the operational level influences other ones.

The answer for the latter question is much easier since we can hardly assume that if a user modifies a query, such modification should anyhow influence the data. However, any of the changes at the logical level can influence the operational level, since the queries may become irrelevant with regard to the data. As discussed in [14], such situation occurs in case the queries do not follow the rules the authors provide. For instance, if a query involves a simple XPath [8] path and any of the elements on the path is deleted, also the path needs to be respectively updated.

6 Propagation Examples

In this section we demonstrate transformation propagation on the transformations introduced in Section 2.

The first transformation was an addition of new child XML element `name` to `purchase-request` in the XML format for purchase requests. We initiate the transformation at the extensional level by modifying an XML document. Since we have added a new element to an XML document, the transformation must be propagated to the logical level and the corresponding XML schema (depicted in Figure 1 on the left) must be extended with the corresponding element declaration. It must be further propagated to the corresponding PSM diagram (depicted in Figure 5 on the left) 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 map the *name* PSM attribute to the exisitng *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 a replacement of XML element `name` with new elements `first-name` and `family-name` in the XML format for sales reports. We initiate this transformation in `SalesReport.xsd`, i.e. at the logical level. Again, the transformation is propagated upwardly first. In the corresponding PSM diagram (depicted in Figure 5 on the right), 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 the PIM diagram must be correspondinly transformed as well, i.e. the attribute *name* of *Customer* must be replaced with new attributes *first-name* and *family-name*. A 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. to the XML format for purchase requests. Since there can be queries for each XML format querying names of customers, they can be influenced as well.

7 Conclusion

The aim of this paper was to show that the problem of XML schema evolution has been highly marginalized so far. In particular, all the existing works only deal with subparts of the problem. Firstly, 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. Secondly, we have discussed how the respective transformations at particular levels influence the neighboring ones. And, finally, 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 want to extend system *XCase* [1] which enables to design conceptual diagrams and map them to respective XML schemas.

Apparently, there exist several open issues: Similarly to the existing works we need to make the adaptation at all levels (and especially the newly proposed ones) efficient and, consequently, we need to be able to find the least expensive sequence of transformations. Secondly, there remains the question of performing the adaptations. Instead of using a brute-force, 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. For the sake of full generality, we intend to involve all the relevant XSD constructs as well as other high-level transformations that we have omitted for simplicity. And, last but not least, we want to extend the approach with concurrency control. 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.

References

1. *XCase – A Tool for XML Data Modeling.* 2008. `http://kocour.ms.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. P. V. Biron and A. Malhotra. *XML Schema Part 2: Datatypes (Second Edition)*. W3C, October 2004. `http://www.w3.org/TR/xmlschema-2/`.
4. S. Boag, D. Chamberlin, M. F. Fernndez, D. Florescu, J. Robie, and J. Simeon. *XQuery 1.0: An XML Query Language*. W3C, January 2007. `http://www.w3.org/TR/xquery/`.
5. B. Bouchou, D. Duarte, M. Halfeld Ferrari Alves, D. Laurent, and M. A. Musicante. Schema Evolution for XML: A Consistency-Preserving Approach. In *Mathematical Foundations of Computer Science*, pages 876–888, Prague, Czech Republic, 2004. Springer-Verlag.
6. T. Bray, J. Paoli, C. M. Sperberg-McQueen, E. Maler, and F. Yergeau. *Extensible Markup Language (XML) 1.0 (Fourth Edition)*. W3C, September 2006. `http://www.w3.org/TR/REC-xml/`.
7. J. Clark. *XSL Transformations (XSLT) Version 1.0*. W3C, November 1999. `http://www.w3.org/TR/xslt`.
8. J. Clark and S. DeRose. *XML Path Language (XPath) Version 1.0*. W3C, November 1999. `http://www.w3.org/TR/xpath`.
9. S. V. Coox. Axiomatization of the evolution of xml database schema. *Program. Comput. Softw.*, 29(3):140–146, 2003.
10. E. Dominguez, J. Lloret, A. L. Rubio, and M. A. Zapata. Evolving XML Schemas and Documents Using UML Class Diagrams. In *DEXA '05: Proceedings of the 16th International Conference on Database and Expert Systems Applications*, pages 343–352, Berlin, Heidelberg, 2005. Springer.
11. M. Klettke. Conceptual XML Schema Evolution - the CoDEX Approach for Design and Redesign. In *BTW Workshops*, pages 53–63. Verlagshaus Mainz, Aachen, 2007.
12. M. Mesiti, R. Celle, M. A. Sorrenti, and G. Guerrini. X-Evolution: A System for XML Schema Evolution and Document Adaptation. In *EDBT '06: Proceedings of the 10th International Conference on Extending Database Technology*, pages 1143–1146, Berlin, Heidelberg, 2006. Springer.
13. I. Mlynkova and M. Necasky. Five-Level Multi-Application Schema Evolution. Technical Report 2008/7, Department of Software Enginneedring, Charles University, Prague, Czech Republic, 2008.
14. M. M. Moro, S. Malaika, and L. Lim. Preserving XML Queries During Schema Evolution. In *WWW '07: Proceedings of the 16th international conference on World Wide Web*, pages 1341–1342, New York, NY, USA, 2007. ACM.
15. M. Necasky. *Conceptual Modeling for XML*. PhD thesis, Charles University, 2008. `http://kocour.ms.mff.cuni.cz/~necasky/dw/thesis.pdf`.
16. H. Su, D. K. Kramer, and E. A. Rundensteiner. XEM: XML Evolution Management. Technical Report WPI-CS-TR-02-09, Computer Science Department, Worcester Polytechnnic Institute, Worcester, Massachusetts, 2002.
17. M. Tan and A. Goh. Keeping Pace with Evolving XML-Based Specifications. In *EDBT '04 Workshops*, pages 280–288, Berlin, Heidelberg, 2005. Springer.
18. H. S. Thompson, D. Beech, M. Maloney, and N. Mendelsohn. *XML Schema Part 1: Structures (Second Edition)*. W3C, October 2004. `http://www.w3.org/TR/xmlschema-1/`.