INFORMATICA, 2015, Vol. 26, No. 3, 453–472
© 2015 *Vilnius University*
DOI: http://dx.doi.org/10.15388/Informatica.2015.58

eXolutio: Methodology for Design and Evolution of XML Schemas Using Conceptual Modeling

Jakub KLÍMEK, Jakub MALÝ, Martin NEČASKÝ, Irena HOLUBOVÁ*
Department of Software Engineering, Faculty of Mathematics and Physics
Charles University in Prague, Czech Republic
e-mail: klimek@ksi.mff.cuni.cz, maly@ksi.mff.cuni.cz, necasky@ksi.mff.cuni.cz, holubova@ksi.mff.cuni.cz

Received: August 2013; accepted: June 2014

Abstract. Recently, XML has achieved the leading role among languages for data representation and, thus, the amount of related technologies and applications exploiting them grows fast. However, only a small percentage of applications is static and remains unchanged since its first deployment. Most of the applications change with newly coming user requirements and changing environment. In this paper we describe a framework and a methodology for management of evolution and change propagation throughout XML applications. We also describe its proof-of-concept implementation called *eXolutio*, which has been developed and improved in our research group during last few years. The text should help the reader to get acquainted with the target area of XML evolution and the approach we have proposed and implemented.

Key words: XML data modeling, evolution of XML applications, change management.

1. Introduction

The eXtensible Markup Language (XML) (Bray *et al.*, 2008) is currently a de-facto standard for data representation and together with accompanying standards, such as XML Schema (Thompson *et al.*, 2004; Biron and Malhotra, 2004), XQuery (Boag *et al.*, 2007), XSLT (Kay, 2007) etc., it becomes a powerful tool for data management. Consequently, the amount and complexity of software systems that utilize XML and/or selected XML-based standards and technologies for information exchange and storage grows very fast. The systems represent information in a form of XML documents. One of the crucial parts of such systems are *XML formats* which describe the syntax of the XML documents. *XML schemas*, expressed in a selected XML schema language, e.g., DTD (Bray *et al.*, 2008) or XSD (Thompson *et al.*, 2004), are used to express the formats. Usually, a system does not use only a single XML format, but a set of different XML formats, each in a particular logical execution part. The XML formats usually represent particular views on the application domain of the software system. For example, a software system for customer relationship management (CRM) exploits different XML formats for purchase orders, customer details, product catalogs, etc. All these XML formats represent different views on

*Corresponding author.

the CRM domain. We can, therefore, speak about a *family of XML formats* used by a software system.

Having such a system, we face the problem of *XML format evolution* as a specific part of evolution of the software system as a whole. The XML formats may need to be evolved whenever user requirements or surrounding environment changes. Each such change may influence multiple different XML formats in the family. Without a proper technique, we have to identify the XML formats affected by the change manually and ensure that they are evolved coherently with each other and the rest of the system. When the XML formats have already been deployed, there will also be XML documents which might become invalid and will require appropriate modification.

In our research group, we have focused on the area of efficient and correct management of a family of XML formats for recent years. Starting with a simple idea of propagation of changes among related XML formats, we have gradually extended our efforts towards a robust framework called *five-level framework for XML design and evolution* and its implementation in a tool called *eXolutio*.[2] It currently supports the original idea of designing XML formats using the principles of *Model Driven Architecture* (MDA) (Miller and Mukerji, 2003), their evolution, and integration of new XML formats.

Contributions. The key contributions of this paper are as follows:

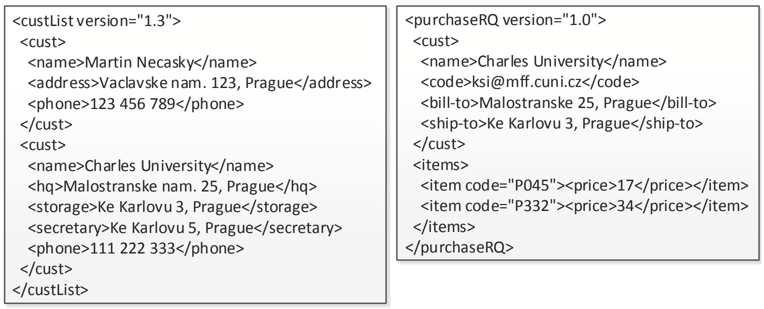
- the description of the five-level framework for design and evolution of a family of XML formats,
- the novel methodology which describes how an XML designer should work with the framework, and
- the evaluation of the methodology and the tool on a complex domain of eHealth proving the concept and efficiency.

Outline. The rest of the paper is structured as follows: In Section 2 we provide a motivation for the problems we solve. In Section 3 we describe the *eXolutio* framework and in Section 4 the methodology for working with the framework. In Section 5 we provide experimental evaluation of our approach. In Section 6 we present a comparative analysis with other tools and in Section 7 we conclude.

2. Motivation

As a demonstration of the problem of evolution and management of XML formats from a more general point of view, let us consider a company that receives purchase orders and let us focus on a part of the system that processes purchases. Let the messages used in the system be XML messages formatted according to a family of different XML formats. Consider the two sample XML documents in Fig. 1. The former one is formatted according to an XML format for a list of customers. The latter one is formatted according to a different

[2]See `http://www.eXolutio.com` for download.



```
<custList version="1.3">
 <cust>
  <name>Martin Necasky</name>
  <address>Vaclavske nam. 123, Prague</address>
  <phone>123 456 789</phone>
 </cust>
 <cust>
  <name>Charles University</name>
  <hq>Malostranske nam. 25, Prague</hq>
  <storage>Ke Karlovu 3, Prague</storage>
  <secretary>Ke Karlovu 5, Prague</secretary>
  <phone>111 222 333</phone>
 </cust>
</custList>
```

```
<purchaseRQ version="1.0">
 <cust>
  <name>Charles University</name>
  <code>ksi@mff.cuni.cz</code>
  <bill-to>Malostranske 25, Prague</bill-to>
  <ship-to>Ke Karlovu 3, Prague</ship-to>
 </cust>
 <items>
  <item code="P045"><price>17</price></item>
  <item code="P332"><price>34</price></item>
 </items>
</purchaseRQ>
```

Fig. 1. XML documents formatted according to different XML schemas.

XML format for purchase requests. There are also other XML formats in the family (e.g., customer details, purchase responses, purchase transport details, etc.). All share the same application domain (customer relationship management). On the other hand, the same part of the domain may be represented according to various XML formats in different ways because of the different purposes for which the XML formats are exploited by the system. For example, the concept of *customer* is represented in each of our sample XML formats in a different way.

Let us now consider a new user requirement that an address should no longer be represented as a simple string. Instead, it should be divided into elements `street`, `city`, `zip`, etc. Such a situation would require a skilled domain expert to identify all the XML formats which involve an address and correct them respectively. In a complex system comprising tens or even hundreds of XML formats (possibly some of them integrated from external partners and, hence, out of our control), this is a difficult and error-prone task. Even identifying the affected parts of an XML format is not an easy and straightforward process. For example, we may need to make the modification only for addresses that represent a place to ship the goods (which are the elements `address` and `storage` in the XML format instantiated in the first schema and element `ship-to` in the second schema). We do not want to modify addresses that represent headquarters, etc. Hence, we need to be able to preserve a kind of semantic relationship between the represented parts of the system.

3. Five-Level Framework for Design, Evolution and Integration of XML Formats

Our five-level evolution management framework allows for design and later maintenance (semantically coherent evolution) of a set of XML formats of a given family. We established a formal base of the framework in Nečaský *et al.* (2011) and firstly described its levels in Nečaský *et al.* (2011a). However, so far we have not described the framework as a whole. We provide such description in this section and we refer to the previous papers for further details. Even so, this section is interesting for the reader familiar with our previous results – it provides a detailed overall description of the framework, verbose explanation of its technical concepts and extensive examples.

The architecture of the framework is depicted in Fig. 2. It is partitioned both horizontally and vertically. Vertical partitions represent individual XML formats. Horizontal

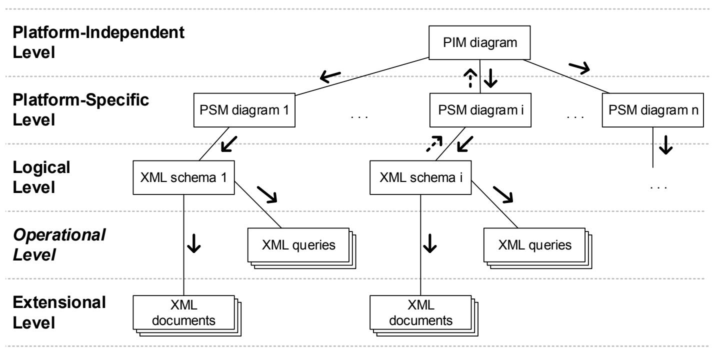


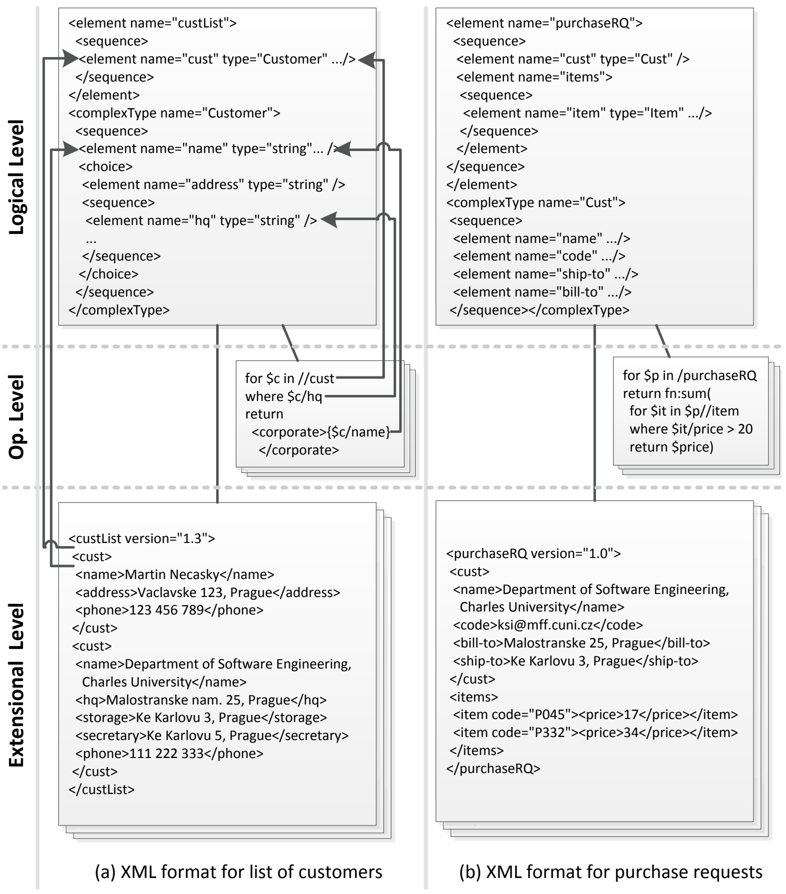
Fig. 2. Five-level XML evolution framework.

partitions represent different levels which characterize each of the XML formats from different viewpoints:

- The *extensional level* contains XML documents formatted according to the XML format.
- The *operational level* contains operations performed over the XML documents from the extensional level. These can be queries over the instances or transformations of the instances.
- The *logical level* contains a logical XML schema which specifies the syntax of the XML format. It is expressed in an XML schema language.
- The *platform-specific level* contains a schema which specifies the semantics of the XML format in terms of the platform-independent level.
- The *platform-independent level* contains a common conceptual schema. It provides the information model of the system and covers the common semantics of the XML formats.

As we can see, the framework covers the syntax and semantics of the XML formats as well as their instances and operations performed over the instances. However, the XML documents, queries and schemas at different horizontal levels are not the only first-class citizens of our framework. There are also mappings between the horizontal levels depicted as solid lines. They are crucial for correct evolution. Evolution means that a change to any XML format made by a designer is correctly propagated to all other relevant affected parts so that all parts of the framework remain consistent. The relevancy results from a particular real-world application. For example, in the area of XML data the propagation from extensional to logical level is not used much, though there exist approaches dealing with this direction (see Section 6). On the other hand, while propagation from logical level to operational level is crucial, the opposite direction makes sense only in very special cases.

As Fig. 2 shows, we consider that the designer makes a change at the logical, platform-specific or platform-independent level (the upward arrows). From here, it is propagated to all other parts of the framework (i.e. to all levels for all XML formats). Changes at


Logical Level
Op. Level
Extensional Level
(a) XML format for list of customers
(b) XML format for purchase requests

```
<element name="custList">
 <sequence>
  <element name="cust" type="Customer" .../>
 </sequence>
</element>
<complexType name="Customer">
 <sequence>
  <element name="name" type="string"... />
  <choice>
   <element name="address" type="string" />
   <sequence>
    <element name="hq" type="string" />
    ...
   </sequence>
  </choice>
 </sequence>
</complexType>
```

```
for $c in //cust
where $c/hq
return
  <corporate>{$c/name}
    </corporate>
```

```
<custList version="1.3">
 <cust>
  <name>Martin Necasky</name>
  <address>Vaclavske 123, Prague</address>
  <phone>123 456 789</phone>
 </cust>
 <cust>
  <name>Department of Software Engineering,
    Charles University</name>
  <hq>Malostranske nam. 25, Prague</hq>
  <storage>Ke Karlovu 3, Prague</storage>
  <secretary>Ke Karlovu 5, Prague</secretary>
  <phone>111 222 333</phone>
 </cust>
</custList>
```

```
<element name="purchaseRQ">
 <sequence>
  <element name="cust" type="Cust" />
  <element name="items">
   <sequence>
    <element name="item" type="Item" .../>
   </sequence>
  </element>
</sequence>
</element>
<complexType name="Cust">
 <sequence>
  <element name="name" .../>
  <element name="code" .../>
  <element name="ship-to" .../>
  <element name="bill-to" .../>
 </sequence></complexType>
```

```
for $p in /purchaseRQ
return fn:sum(
  for $it in $p//item
  where $it/price > 20
  return $price)
```

```
<purchaseRQ version="1.0">
 <cust>
  <name>Department of Software Engineering,
    Charles University</name>
  <code>ksi@mff.cuni.cz</code>
  <bill-to>Malostranske 25, Prague</bill-to>
  <ship-to>Ke Karlovu 3, Prague</ship-to>
 </cust>
 <items>
  <item code="P045"><price>17</price></item>
  <item code="P332"><price>34</price></item>
 </items>
</purchaseRQ>
```

Fig. 3. Sample XML formats represented in the framework.

the operational or extensional level can also appear. However, as we have mentioned, even though it could be theoretically possible to propagate them to the upper levels, it is usually not meaningful. For example, it is not much meaningful to propagate a change from the extensional level (i.e. a change made in a particular XML document) to the logical level (i.e. to the corresponding XML schema) and higher. A change in an XML document does not usually mean that the respective XML schema needs to be adapted. On the other hand, the framework can identify that the change in the XML document does not correspond to the XML schema and notify the designer. However, we omit this issue in this paper and focus on the more "natural" cases.

3.1. *Mapping Between the Levels*

The lower three levels are depicted in Fig. 3, representing our two sample XML formats from Fig. 1. If we consider only them, we have no explicit relationship between the vertical partitions, i.e. between the XML formats modeled by the framework. As we have already discussed, a change in one XML format can trigger changes in the other XML formats to keep their consistency. Therefore, a change in one XML schema must be propagated to the other affected XML formats manually by a designer. This is, of course, highly time-consuming and error-prone solution. The designer must be able to identify all the affected formats and propagate the change correctly. Often, (s)he is not able to do such a complex work and needs a help of a domain expert who understands the problem domain, but is, typically, a business expert rather than a technical XML format expert. Therefore, it is very hard for him to navigate in the logical XML schemas, operations and instances.

To overcome these problems, we introduce the two additional levels. They represent two additional levels of abstraction of the XML formats and are motivated by the MDA (Miller and Mukerji, 2003) principles. The *platform-independent level* comprises a single conceptual schema of the problem domain. We call it *PIM schema* and use the notation of UML (2007) class diagrams to express it. A sample PIM schema modeling the domain of customers and their purchases is depicted in Fig. 4. The *platform-specific level* comprises an individual schema for each XML format. We call it *PSM schema* and use UML class diagrams as well. We introduced few extensions to the UML notation to be suitable for modeling XML formats, so-called *XSEM schema* (Nečaský, 2009). However, these extensions are not important for this paper. For their full description we refer to Nečaský *et al.* (2011). A PSM schema has a hierarchical structure since it models and XML format. Two sample PSM schemas for our two XML formats are depicted in Fig. 4.

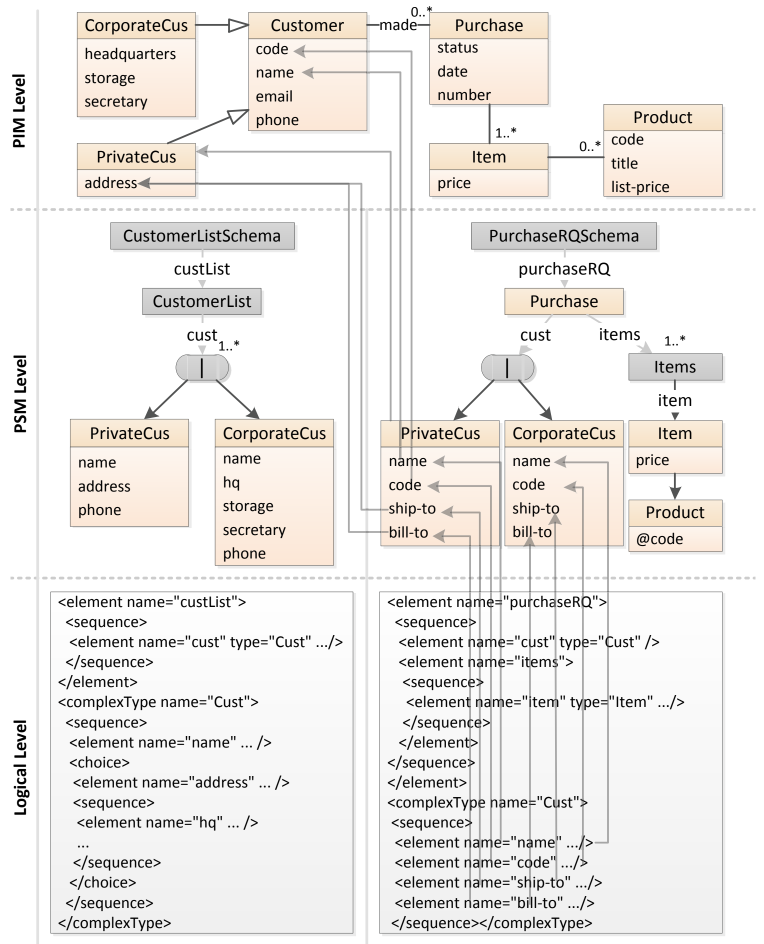
4. Methodology for Design and Evolution of XML Formats

In this section, we introduce a methodology which guides XML format designers in using our framework to (1) design new XML formats which are semantically consistent with already existing XML formats in the family, (2) integrate existing XML formats into the framework (e.g., XML formats defined by an industrial standardization organization), and (3) evolve the whole family of XML formats while preserving the achieved consistency.

The methodology consists of various steps. Some of them must be done *manually* by the designer (and/or domain expert). Some of them are performed *automatically* by the framework or *semi-automatically*, i.e. the framework finds possible solutions and a human user selects the correct one. In this section, we do not describe the algorithms for the automatic and semi-automatic steps. They are thoroughly described in our previous paper (Nečaský *et al.*, 2011a). We only point out what needs to be done manually and what can be automated. The methodology is described in steps performed by the designer and/or system. We denote the steps which must be performed manually by the designer with **(M)**. The steps which are performed automatically by the system are marked with **(A)**. The steps which need the system to cooperate with the designer are denoted with **(S–A)**.

4.1. *Forward-Engineering of XML Formats*

Let us first discuss a methodology which allows a designer to design a new XML format. The designer proceeds in the forward direction from the PIM level to the logical level and, therefore, we call the methodology *forward-engineering of XML schemas*. The result of



```
<element name="custList">
 <sequence>
  <element name="cust" type="Cust" .../>
 </sequence>
</element>
<complexType name="Cust">
 <sequence>
  <element name="name" ... />
  <choice>
   <element name="address" ... />
   <sequence>
    <element name="hq" ... />
    ...
   </sequence>
  </choice>
 </sequence>
</complexType>
```

(a) XML format for list of customers

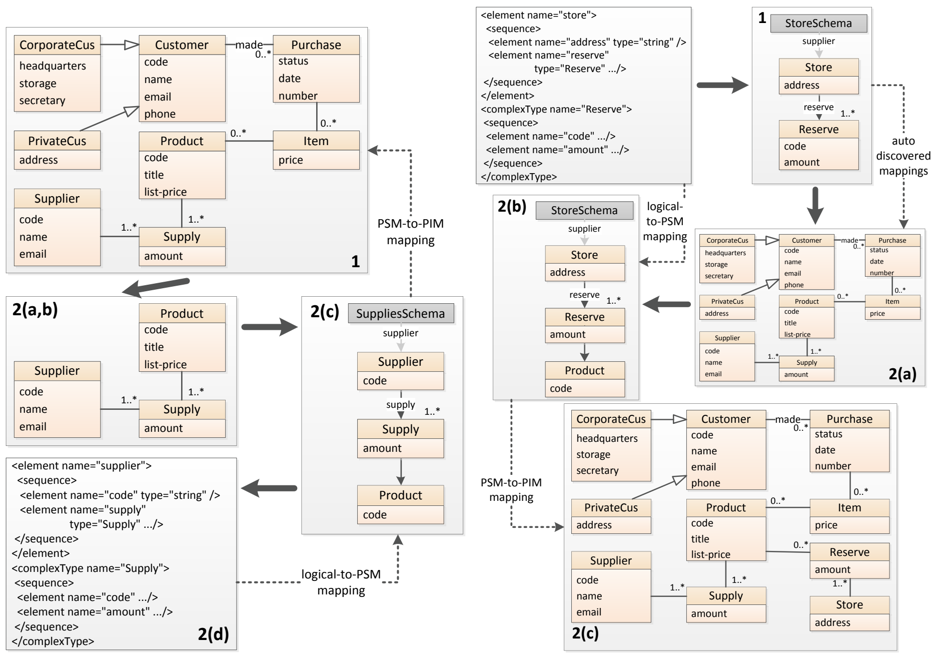
```
<element name="purchaseRQ">
 <sequence>
  <element name="cust" type="Cust" />
  <element name="items">
   <sequence>
    <element name="item" type="Item" .../>
   </sequence>
  </element>
 </sequence>
</element>
<complexType name="Cust">
 <sequence>
  <element name="name" .../>
  <element name="code" .../>
  <element name="ship-to" .../>
  <element name="bill-to" .../>
 </sequence></complexType>
```

(b) XML format for purchase requests

Fig. 4. Sample XML formats represented at logical, PSM and PIM levels.

the process are XML schema, PSM schema and possible extension to the PIM schema and also mappings between them. We demonstrate the methodology in Fig. 5(a). Here, a designer is given a task to design a new XML format for a list of supplies of a given supplier. The designer initiates the following steps:

1. **(M)** The application domain is studied and described in a form of a PIM schema. It may happen that the PIM schema already exists but it does not fully cover the semantics of the designed XML format. Hence, it must be extended. The designer cooperates with a domain expert. This is necessarily a *manual* process. In our sample scenario, the designer extends the PIM schema with the model of suppliers (class `Supplier`) and product supplies (class `Supply`) (Fig. 5(a), step 1).
2. For each XML format which needs to be newly designed:
   (a) **(M)** In cooperation with the domain expert, the designer analyzes what information (i.e. relevant concepts and relationships) must be represented in the in-



```
<element name="supplier">
 <sequence>
  <element name="code" type="string" />
  <element name="supply"
            type="Supply" .../>
 </sequence>
</element>
<complexType name="Supply">
 <sequence>
  <element name="code" .../>
  <element name="amount" .../>
 </sequence>
</complexType>
```

```
<element name="store">
 <sequence>
  <element name="address" type="string" />
  <element name="reserve"
            type="Reserve" .../>
 </sequence>
</element>
<complexType name="Reserve">
 <sequence>
  <element name="code" .../>
  <element name="amount" .../>
 </sequence>
</complexType>
```

(a) Forward-engineering (b) Reverse-engineering

Fig. 5. Demonstrations of forward/reverse engineering methodologies.

stances of the target XML format (Fig. 5(a), step 2(a)). This process is necessarily *manual*.

(b) **(M)** The designer *manually* identifies the part of the PIM schema which models the concepts and relationships identified in the previous step. In our example, it means classes `Product`, `Supply` and `Supplier` (Fig. 5(a), step 2(b)).

(c) **(S–A)** The selected part of the PIM schema is shaped into a PSM schema which models the aimed XML schema. The conversion is *partly manual and partly automatic*. The designer specifies the hierarchical structure manually. The mappings to the PIM schema are generated *automatically* as the designer builds the PSM schema from the PIM schema part. The designer must *manually* create additional PSM components which are not mapped to the PIM schema. In our example, the designer specifies that class `Supplier` will be specified as a root with nested class `Supply` which contains a product code (attribute `code` of class `Product`) and a supplied amount (attribute `amount`) (Fig. 5(a), step 2(c)).

(d) **(A)** The resulting PSM schema is *automatically* converted to a logical XML schema expressed in the selected XML schema language. The mapping of the logical XML schema to the PSM schema is also created *automatically* (Fig. 5(a), step 2(d)). The full translation algorithm was published in Nečaský *et al.* (2011).

The methodology has several advantages. The designer works at the user-friendly level of UML class diagrams abstracted from the details of XML schemas. It also saves time, because shaping the part of the PIM schema to the PSM schema is much easier than manual creation of the XML schema. And, it avoids errors since the overall description of the problem domain is given and, therefore, the designer does not miss important real-world concepts or relationships in the designed XML schema. Additionally, (s)he does not add new concepts without seeing the impact to the overall PIM schema, which avoids introducing redundant or overlapping structures into the PIM.

4.2. *Reverse-Engineering of XML Formats*

The designer can also integrate existing XML schemas into his/her solution in the *bottom-up* manner which we also call *reverse-engineering*. Such XML schema might be a legacy XML schema or an XML schema prescribed by a standardization organization or used by other party that we want to integrate into our system. We demonstrate the reverse-engineering methodology in a sample scenario depicted in Fig. 5(b). A designer is given a task to integrate an existing XML schema of an XML format for reserves on a given store. The designer proceeds in the following steps:

1. **(A)** The XML schema is *automatically* converted to a corresponding PSM schema. The mapping of the XML schema to the PSM schema is also created automatically (Fig. 5(b), step 1). The full translation algorithm was published in Nečaský *et al.* (2011).
2. The designer maps the PSM schema to the PIM schema in the following steps:
   (a) **(A)** Various algorithms for *automatic* mapping discovery (Euzenat and Shvaiko, 2007) are applied to help the designer with creating the mappings (Fig. 5(b), step 2(a)).
   (b) **(M)** The designer can be required to augment the generated PSM schema *manually*. In our case, the generated class `Reserve` contains attributes `code` and `amount`. However, `code` belongs to PIM class `Product` from the conceptual perspective while `amount` belongs to the reserve information which is not modeled in the PIM schema yet. Therefore, the designer moves `code` to PSM class `Product` which will be mapped to PIM class `Product`. (S)he also creates an association connecting PSM classes `Reserve` and `Product` (Fig. 5(b), step 2(b)).
   (c) **(M)** (S)he might also be required to *manually* complement the PIM schema in case it does not cover the semantics of the imported XML schema. The impact of the performed changes in the PIM schema to the other XML formats integrated in the framework is analyzed and reported to the designer. In our case, the designer creates new PIM classes `Reserve` and `Store` to cover the new kind of information (Fig. 5(b), step 2(c)).

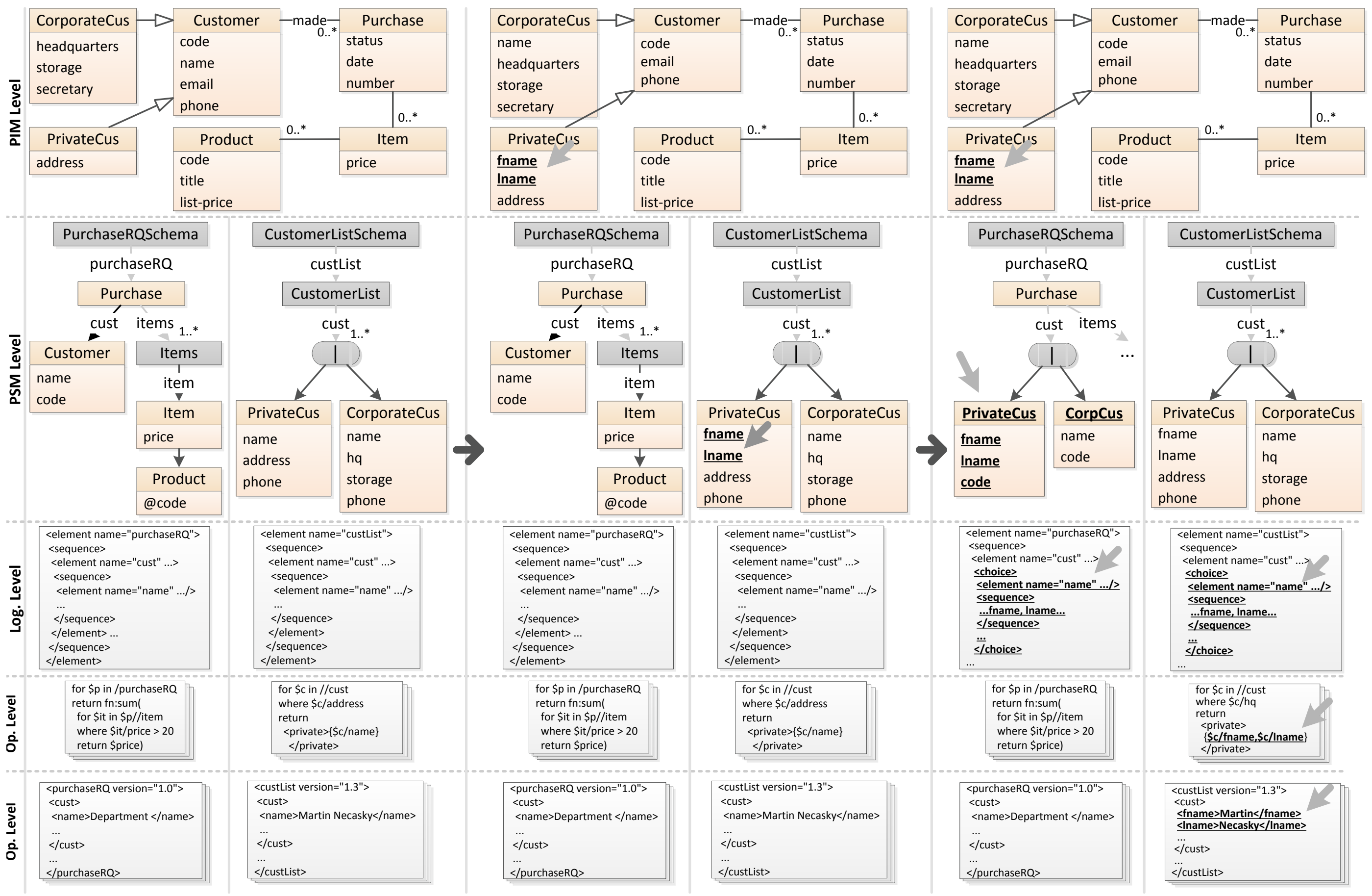
The advantages are similar to the forward-engineering methodology. The main advantage for the designer is that (s)he works with UML class diagrams. It is much easier to map the PSM schema than the XML schema (because both PSM and PIM schema uses the UML notation).

4.3. *Evolution of XML Formats*

When all required XML formats are designed or integrated into the framework it is in some point in time necessary to evolve them consistently. We demonstrate the evolution methodology in a sample scenario depicted in Fig. 6. The designer needs to split private customer single-valued name into two values first name and last name in the PSM schema of the XML format for customer lists (the right-hand side PSM schema in Fig. 6(a)). The scenario shows how the change is propagated to all affected parts of the framework. The designer proceeds in the following steps:

1. **(M)** The designer analyzes a new user requirement or change in the system environment in cooperation with the domain expert. This is necessarily a *manual* process. (S)he decides whether the subject of change is the whole application domain or only a particular XML format. In the former case, it is necessary to implement the change at the PIM level. In the latter one, the change will be done at the PSM level. The designer can also make a change in a particular XML document, i.e. at the extensional level.
2. **(M)** The designer makes the initial change at the identified level. In our sample scenario, the initial change was made in the PSM schema on the right-hand side of Fig. 6(b) as indicated by the black arrow pointing to class `PrivateCus` with two new attributes `fname` and `lname` which replaced the original attribute `name`.
3. First, the upwards propagation is performed:
   (a) **(A)** The impact analysis of the change to the upper level is computed *automatically* and presented to the designer (see Nečaský *et al.*, 2011a for formal description of impact analysis algorithms). The mappings between the levels are exploited. The analysis shows inconsistencies caused by the change. The designer can rollback the initial change and the whole propagation process at this point.
   (b) **(S–A)** Possible scenarios of propagation of the change to the upper level are identified *automatically* and proposed to the designer (see Nečaský *et al.*, 2011a for formal description of impact analysis algorithms). Each scenario consists of a sequence of change operations which needs to be performed at the upper level to restore the consistency between the current and upper level. The designer selects one of the scenarios *manually*.
   (c) **(A)** The chosen scenario is performed at the upper level automatically.
   (d) If the PIM level is not reached, steps 3(a–c) are repeated for each operation from the chosen scenario. The upper level becomes the current level.

   In our sample scenario, the initial change was performed at the PSM level, so the upwards propagation has only one cycle – propagation to the PIM level. The result is highlighted in Fig. 6(b) by the grey arrow pointing to PIM class `PrivateCus` with two new attributes `fname` and `lname` instead of the original attribute `name`.
4. Second, the downwards propagation is performed for each operation made at the PIM level. For each XML format, the propagation continues until the extensional level is reached:



```
<element name="purchaseRQ">
 <sequence>
  <element name="cust" ...>
   <sequence>
    <element name="name" .../>
    ...
   </sequence>
  </element> ...
 </sequence>
</element>
```

```
<element name="custList">
 <sequence>
  <element name="cust" ...>
   <sequence>
    <element name="name" .../>
    ...
   </sequence>
  </element>
 </sequence>
</element>
```

```
for $p in /purchaseRQ
return fn:sum(
 for $it in $p//item
 where $it/price > 20
 return $price)
```

```
for $c in //cust
where $c/address
return
 <private>{$c/name}
  </private>
```

```
<purchaseRQ version="1.0">
 <cust>
  <name>Department </name>
  ...
 </cust>
 ...
</purchaseRQ>
```

```
<custList version="1.3">
 <cust>
  <name>Martin Necasky</name>
  ...
 </cust>
 ...
</custList>
```

```
<element name="purchaseRQ">
 <sequence>
  <element name="cust" ...>
   <choice>
    <element name="name" .../>
    <sequence>
     ...fname, lname...
    </sequence>
    ...
   </choice>
 ...
```

```
<element name="custList">
 <sequence>
  <element name="cust" ...>
   <choice>
    <element name="name" .../>
    <sequence>
     ...fname, lname...
    </sequence>
    ...
   </choice>
 ...
```

```
for $c in //cust
where $c/hq
return
  <private>
   {$c/fname,$c/lname}
  </private>
```

```
<custList version="1.3">
 <cust>
  <fname>Martin</fname>
  <lname>Necasky</lname>
  ...
 </cust>
 ...
</custList>
```

Fig. 6. Demonstration of evolution methodology.

(a) **(A)** The impact analysis of the change to the lower level is computed automatically and presented to the designer. The mappings between the levels are exploited. Similarly to step 3(a), the designer can rollback the whole propagation process here.
(b) **(S–A)** Similarly to step 3(b), possible scenarios of propagation of the change to the lower level are identified and the designer selects one of them *manually*.
(c) **(A)** The chosen scenario is performed at the lower level *automatically*.
(d) If the extensional level is not reached, steps 4(a)–4(c) are repeated for each operation from the chosen scenario. The lower level becomes the current level.

In our sample scenario, the downwards propagation has an impact on both XML formats as highlighted in Fig. 6(c) by the grey arrow.

The approach naturally preserves the semantic consistency between the XML formats. It facilitates the work of the designer by performing the impact analysis first and then by providing possible propagation scenarios. The designer only selects from the offered possibilities or provides own scenarios when the offered ones are not sufficient or none scenario can be offered. The propagation according to the selected scenario is then automatic.

5. Experiments

We made experimental evaluation of the proposed approach in the domain of electronic health (eHealth). We experimented with the *Data Standard for eHealth in the Czech Republic* (DASTA).[3] The data standard comprises tens of XML formats for exchanging information about patients, drugs, hospitals, medical examinations, etc. Only XML schemas and textual documentation is provided by the DASTA authors. We therefore extended DASTA with a PIM schema of the medical domain and modeled 14 XML formats with PSM schemas. To evaluate the methodologies presented in this paper, we created some of the PSM schemas using the forward engineering methodology and others using the reverse engineering methodology. We then proceeded with the evaluation of the evolution methodology.

To be able to do the evaluation, our first goal was to design the PIM schema on the basis of the textual documentation provided by the DASTA authors. Its part is shown in Fig. 7. It models patients and doctors, insurance, diagnoses, vaccination and working disabilities. The whole PIM schema contains 76 classes, 124 associations and more than 300 of attributes.

We first evaluated our *forward-engineering methodology*. We chose 4 different XML formats for representing various kinds of XML messages related to manipulation with drugs (prescription, prescribed drugs, dispensation and vaccination detail). We did the experiment with 2 XML schema designers. Each created PSM schemas of two of the XML formats from the provided PIM schema. The resulting PSM schemas contain 42 classes, 38 associations and 76 attributes in total.

[3] `http://ciselniky.dasta.mzcr.cz/` (in Czech).

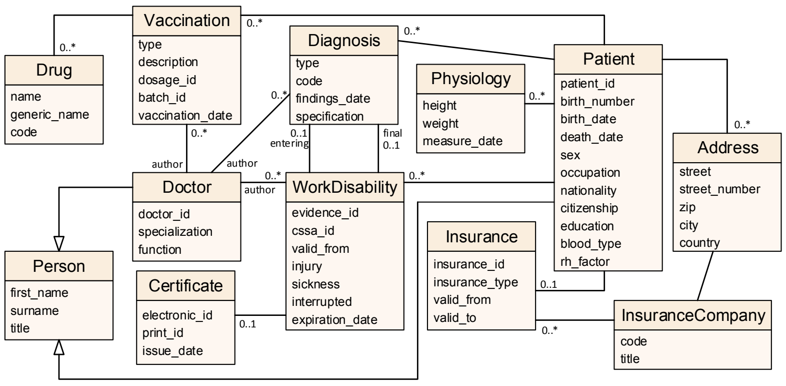


Fig. 7. PIM schema of eHealth domain in the Czech Republic.

We discussed the results with the designers as well as with the authors of DASTA. Their experience and evaluation is summarized below:

1. A designer saves a significant amount of time. The 4 XML schemas contain 106 declarations of XML elements and attributes (the declarations are modeled by all attributes and some associations). Instead of encoding them manually, which takes a lot of coding time, the designer just does drag and drop operations. He drags and drops classes and associations from the pre-defined PIM schema to the PSM schemas (i.e. $42 + 38$ drag and drop operations). For each of the 42 classes he chooses required PIM attributes from an automatically displayed list. These are automatically moved to the PSM schema.
2. The number of errors in the modeled XML schemas and semantic inconsistencies among them caused by different understandings of the domain by the XML schema developers is reduced for each of the 156 PIM components thanks to the common PIM schema.
3. The accuracy of the resulting XML schemas can be easily checked by a domain expert since he can validate each declaration in the XML schemas against the PIM schema visually without studying complex and technical XML schema code. The tool automatically shows a mapping of each chosen declaration to the corresponding PIM component. Therefore, the time needed to find the corresponding PIM component is significantly reduced.
4. It can be easily checked whether any of more than 500 of medicinal concepts modeled in the whole PIM schema is represented in the XML schemas and how. Vice versa, having one of the declarations from the resulting XML schemas it is easy to see in the PIM schema what medicinal concept it represents.

Even though it is hard to exactly measure the time saved and errors reduced, the experiment demonstrates that the savings are significant. The time the designer needs to manually encode an XML element or attribute declaration is much longer than the time required to dragging and dropping components from the PIM schema to the PSM schemas

. Even more time is saved when the domain expert checks the validity of the created XML schemas against the conceptual representation expressed in the form of the PIM schema.

We also evaluated our *reverse-engineering methodology*. We chose 8 XML schemas from the rest of the DASTA XML schemas (i.e. from those we did not use for the forward engineering evaluation). According to the methodology, we converted them to their PSM equivalents. This is an automatic process without participation of the designer. The resulting PSM schemas contained 62 classes, 54 associations and 93 attributes. Most of them were then mapped to their PIM equivalents automatically. This is not surprising because the PIM schema was created manually according to the XML schemas. Therefore, the names of the XML elements and attributes declared in the XML schemas correspond to the names of classes, attributes and associations in the PIM schema. If the reader is interested in how our reverse engineering method is successful in cases where the names in the PSM schemas do not exactly correspond to the PIM schema we refer to Klímek and Nečaský (2010).

There were several inconsistencies which could not be resolved automatically and the mapping had to be created manually. This included approx. 20% of PSM components. The revealed inconsistencies resulted in proposals of changes to the original XML schemas so that they could be made semantically consistent with the rest of the DASTA XML schemas. For example, it has appeared that 3 different kinds of medical doctor identifiers were used in three different XML schemas. We discussed the PIM and PSM schemas with the DASTA authors and they discovered other inconsistencies which they did not see while reading the XML schemas (different structures for addresses, identifiers, etc.).

The evaluation shows that the reverse engineering process helps to reveal various errors and semantical inconsistencies among XML schemas in a given set. The inconsistencies are discovered mainly during the mapping of automatically generated PSM schemas to the PIM schema. The components which cannot be mapped automatically are candidate inconsistencies and need to be inspected by the designer. Most of them are hard to reveal when only XML schemas exist which need to be explored manually. On the other hand, our approach requires the existence of the PIM schema – its creation takes some time.

And, finally, we also evaluated the *evolution methodology*. The DASTA standard is changed four times a year by the authors. This makes 24 versions since 2006. We do not discuss all the changes which appeared in these 24 versions. Instead we select one of the recent versions which included an interesting change – replacement of one of the original DASTA XML formats related to working disabilities with another XML format for the same kind of messages used by the *Czech Social Security Administration* (CSSA).[4] In this experiment, we evolved the PSM schema of the original XML format to a new version which modeled the XML format used by CSSA. The old one is depicted in Fig. 8(a). The new one is depicted in Fig. 8(b). The figures contain only parts of the PSM schemas – the original PSM schemas are more complex (they contain 10/29 classes, 9/28 associations, and 14/41 attributes, respectively).

There are many differences between both versions. Some of them are displayed in Fig. 8 in red. For example, the red arrow connecting the attribute *interruption* on the left

[4] `http://www.cssz.cz/en/about-cssa/`.

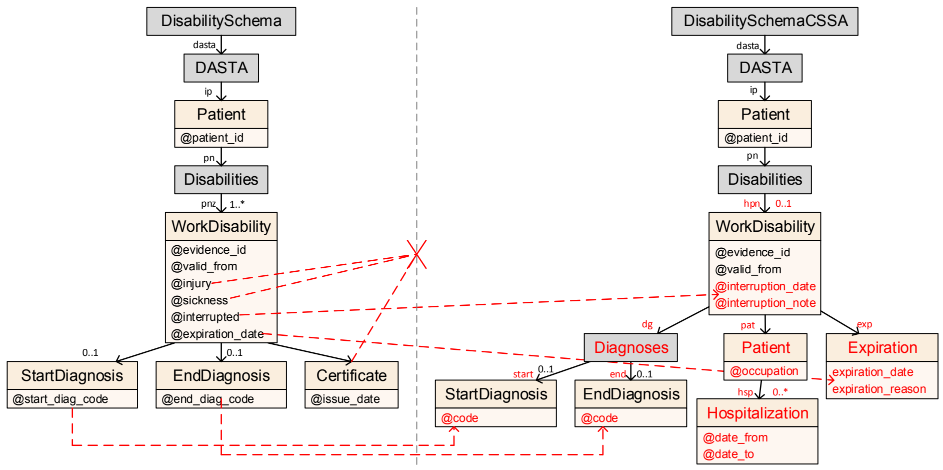


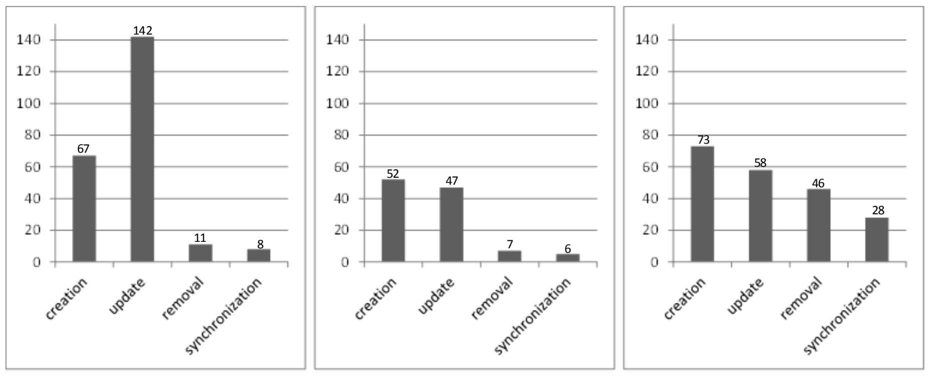
Fig. 8. PSM schema of (a) old XML format for working disabilities and (b) CSSA XML format for working disabilities.

with the attributes *interruption_date* and *interruption_note* on the right show that the original attribute was split by the designer to 2 new attributes. On the other hand, the attribute *injury* has been removed. Some components were renamed, e.g., *start_diag_code* and *end_diag_code* were both renamed to *code*. Some components have been newly created, e.g., class *Hospitalization* on the right.

Many changes made at the PSM level needed to be propagated to the PIM schema because they affected the conceptual representation of the medical domain. For example, the split of the attribute *interruption* was propagated in this way. From here, they needed to be propagated to the other PSM schemas. (For the experiment, we had 12 PSM schemas modeled from the previous evaluation of the forward and reverse engineering methodologies where the changes had to be propagated from the PIM level.) The designer performs edit operations on a chosen schema. The edit operations are expressed as a sequence of atomic operations of four kinds: creation, removal, update and synchronization. The designer does not use these atomic operations as they are too simple. Instead, he performs operations like *split attribute* which is defined as a sequence of atomic operations of all kinds. Therefore, the designer performs a single operation but, in fact, he executes a sequence of several atomic operations.

In the experiment, we measure the number of atomic operations which had to be performed to evolve the old PSM schema to the new one. We also measure the number of atomic operations which were performed by our propagation mechanism to ensure that the PIM schema and all related PSM schemas are evolved correspondingly. We show the total number of atomic operations, the number of operations which were performed by the designer manually, and the number of operations which were performed automatically by our propagation mechanism.

The result is depicted in Fig. 9. Figure 9(a) shows the number of operations performed manually by the designer to evolve the old version of the PSM schema to the new ver-



(a) Manually executed operations (b) Operations automatically propagated from PSM to PIM level (c) Operations automatically propagated from PIM to PSM level

Fig. 9. Numbers of atomic operations performed during the evolution of the XML format depicted in Fig. 8.

sion. Figure 9(b) shows the number of operations *automatically* propagated to the PIM schema on the basis of the manually performed operations from Fig. 9(a). Figure 9(c) shows the number of operations *automatically* propagated from the PIM level to all other PSM schemas. The columns of each graph represent the kinds of operations – creation, update, removal, synchronization. For example, Fig. 9(a) says that there were 67 creation operations, 142 updates, etc.

Let us now analyze the numbers of operations in a more detail. The numbers show that there have been 67 creation operations performed manually by the designer. However, the propagation mechanism resulted only in 52 creation operations performed automatically on the PIM schema. This is because not all creation operations at the PSM level lead to creation operations at the PIM level. The designer may create a PSM component which only models a grammatical rule without a semantical equivalent at the PIM level. The same holds for the other kinds of operations. The most significant difference is in update operations. Only 1/3 of update operations were propagated to the PIM level. This is because the designer only renamed XML components without changing the names of their PIM equivalents. In case of removal and synchronization operations, the reason is that removed or synchronized components have no equivalent at the PIM level.

Figure 9(b) demonstrates the amount of work saved by our propagation mechanism to keep the PIM schema consistent with changes made by the designer in the PSM schema. Without propagation, the designer would have to perform all operations summarized in Fig. 9(b) manually. If we compare the numbers with the numbers in Fig. 9(a) we see that we saved more than 1/3 of operations. Figure 9(c) shows the number of operations performed automatically by our propagation mechanism on other PSM schemas after the PIM schema has been automatically updated. These operations are necessary to keep the other PSM schemas semantically consistent with the changes made by the designer in the primary updated PSM schema.

The numbers show that without our propagation mechanism the designer would have to do a lot of operations manually on other XML schemas to keep them semantically

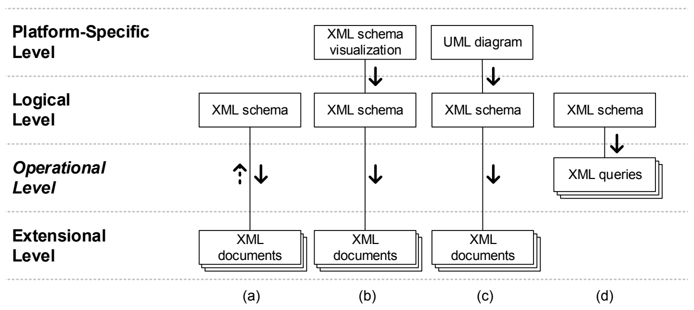


Fig. 10. Other existing approaches in scope of the framework.

consistent. More than 1/2 of the necessary operations were performed automatically by our change propagation mechanism. However, this means saving much more than 1/2 of time. For each change in the primary XML schema the designer would have to think how to propagate the change to the other XML schemas. Using our approach the designer works at the PSM level and the changes are propagated automatically through the PIM schema.

6. Comparative Analysis

Naturally, *eXolutio* is not the first tool that deals with change management of XML applications. However, none of the existing approaches focuses on such a complex case, or they solve only a part of the problem.

We can divide the current approaches into several groups depicted in the context of our framework in Fig. 10. Approaches in the first group (Fig. 10(a)) consider changes at the logical level and differ in the selected XML schema language, i.e. DTD (Al-Jadir and El-Moukaddem, 2003; Coox, 2003) or XML Schema (Tan and Goh, 2005; Cavalieri, 2010). In general, the transformations can be variously classified. For instance, paper of Tan and Goh (2005) distinguishes *migratory* (e.g., movements of elements/attributes), *structural* (e.g., adding/removal of elements/attributes) and *sedentary* (e.g., modifications of simple data types). The changes are expressed variously and more or less formally. For instance in Cavalieri (2010) a language called *XSUpdate* is described. The changes are then automatically propagated to the extensional level to ensure validity of XML data. There also exists an opposite approach that enables one to evolve XML documents and propagate the changes to their XML schema (Bouchou *et al.*, 2004). Approaches in the second and third group (Fig. 10(b) and 10(c)) are similar, but they consider changes at an abstraction of logical level – either visualization (Klettke, 2007) or a kind of UML diagram (Domínguez *et al.*, 2005). Both work at the PSM level, since they directly model XML schemas with their abstraction. However, no PIM schema is considered, since all the three groups consider only a single evolving XML schema.

Another open problem related to schema evolution is adaptation of the respective XML queries (see Fig. 10(d)), i.e. propagation to the operational level. Unfortunately, the amount of existing works is relatively low. Paper of Moro *et al.* (2007) gives recommendations on how to write queries that do not need to be adapted for an evolving schema. In Geneves *et al.* (2009) the authors consider a subset of XPath 1.0 for which they study the impact changes in an XML schema.

Contrary to the described papers, in Passi *et al.* (2009) multiple *local* XML schemas are considered and mapped to a *global* object-oriented schema. However, the global schema does not represent a common problem domain, but a common integrated schema; the changes are propagated just upwards and the operations are not defined rigorously. The need for a well defined set of simple operations and their combination is identified in Bellahsene *et al.* (2011).

In the field of commercial tools, every tool profiles itself either as a UML tool (EA, 2014; MagicDraw, 2014) or an XML tool (Oxygen, 2014; XML Spy, 2014). Some of the UML tools offer features for translation UML class diagrams to XML schemas (EA, 2014), but the translation algorithms are straightforward and not applicable for the situation where a family of schemas describe the common model. In the state-of-the-art XML tools, the user can use various views, representations and visualizations for editing, querying and validating XML schemas and documents, but they do not target schema evolution. Tools for XML schema comparison are available as well (Coox, 2003). They operate at the logical level and solving the ambiguity (required in all comparison approaches) is left up to the user. None of the XML tools, however, offers mapping a family of schemas to a common model and complex evolution with propagation, as does *eXolutio*.

7. Conclusion and Open Problems

The aim of this paper was to provide a summary and overview of the work that has been done in our research group in the area of XML evolution in recent years. We have described various aspects of our general framework, called *eXolutio*, which serves as an experimental tool for proving the proposed concepts and evaluation of respective algorithms. This text should help the reader to get acquainted with the target area and to find the relation between the published papers and our work.

In our future work we want to continue in our effort towards a robust evolution management framework and extend it so that it can be used in the industry as a complete tool. We also want to focus on several open issues, namely propagation of changes to the operational level, preserving of integrity constraints at all the levels and also other data formats than XML.

Acknowledgements. This work was supported by the project SVV-2014-260100.

References

Al-Jadir, L., El-Moukaddem, F. (2003). Once upon a time a DTD evolved into another DTD... In: *Proceedings of OOIS 2003*. Springer, Berlin/Heidelberg, pp. 3–17.

Bellahsene, Z., Bonifati, A., Rahm, E. (2011). *Schema Matching and Mapping, Section 6. Data-Centric Systems and Applications*. Springer, Berlin/Heidelberg.

Biron, P.V., Malhotra, A. (2004). *XML Schema Part 2: Datatypes*, second edition. W3C. http://www.w3.org/TR/xmlschema-2/.

Boag, S., Chamberlin, D., Fernández, M.F., Florescu, D., Robie, J., Siméon, J. (2007). *XQuery 1.0: An XML Query Language*. W3C. http://www.w3.org/XML/Query/.

Bouchou, B., Duarte, D., Halfeld, M., Alves, F., Laurent, D., Musicante, M.A. (2004). Schema Evolution for XML: A Consistency-Preserving Approach. *Mathematical Foundations of Computer Science*. Springer, Prague, pp. 876–888.

Bray, T., Paoli, J., Sperberg-McQueen, C.M., Maler, E., Yergeau, F. (2008). *Extensible Markup Language (XML) 1.0*, fifth edition. W3C. http://www.w3.org/XML/.

Cavalieri, F. (2010). EXup: an engine for the evolution of XML schemas and associated documents. In: *Proceedings of 2010 EDBT/ICDT Workshops*. ACM Press, New York, pp. 1–10.

Coox, S. V. (2003). Axiomatization of the evolution of XML database schema. *Programming and Computer Software*, 29(3), 140–146.

DiffDog (2014). *Altova*. http://www.altova.com/diffdog/.

Domínguez, E., Lloret, J., Rubio, A.L., Zapata, M.A. (2005). Evolving XML schemas and documents using UML class diagrams. In: *Proceedings of DEXA 2005*, *LNCS*, Vol. 3588. Springer, Berlin, pp. 343–352.

EA (2014). *Enterprise Architect*. http://www.sparxsystems.com.au/products/ea/index.html.

Euzenat, J., Shvaiko, P. (2007). *Ontology Matching*. Springer, Berlin, Heidelberg. http://book.ontologymatching.org/.

Geneves, P., Layaida, N., Quint, V. (2009). Identifying query incompatibilities with evolving XML schemas. In: *Proceedings of ICFP 2009*, ACM Press, New York, pp. 221–230.

Kay, M. (2007). *XSL Transformations (XSLT) Version 2.0*. W3C. http://www.w3.org/TR/xslt20/.

Klettke, M. (2007). Conceptual XML schema evolution – the CoDEX approach for design and redesign. In: *Proceedings of BTW 2007 Workshop*, Aachen, Germany, pp. 53–63.

Klímek, J., Nečaský, M. (2010). Integrating XML schemas for evolution of web services. In: *Proceedings of ICWS 2010*, IEEE Computer Society, Miami, pp. 307–314.

MagicDraw (2014). http://www.magicdraw.com/.

Miller, J., Mukerji, J. (2003). *MDA Guide Version 1.0.1*. Object Management Group. http://www.omg.org/cgi-bin/doc?omg/03-06-01.

Moro, M.M., Malaika, S., Lim, L. (2007). Preserving XML queries during schema evolution. In: *Proceedings of WWW 2007*, ACM Press, New York, pp. 1341–1342.

Nečaský, M. (2009). *Conceptual Modeling for XML. Dissertations in Database and Information Systems*, Vol. 99. IOS Press, Amsterdam.

Nečaský, M., Klímek, J., Malý, J., Mlýnková, I. (2011a). Evolution and change management of XML-based systems. *Systems and Software*, 85(3), 683–707.

Nečaský, M., Mlýnková, I., Klímek, J., Malý, J. (2011b). When conceptual model meets grammar: a dual approach to XML data modeling. *Data & Knowledge Engineering*, 72, 1–30.

Oxygen (2014). *An XML Editor*. http://www.oxygenxml.com.

Passi, K., Morgan, D., Madria, S. (2009). Maintaining integrated XML schema. In: *Proceedings of IDEAS 2009*, ACM Press, New York, pp. 267–274.

Tan, M., Goh, A. (2005). Keeping pace with evolving XML-based specifications. In: *Proceedings of EDBT 2004 Workshops*, Springer, Berlin/Heidelberg, pp. 280–288.

Thompson, H.S., Beech, D., Maloney, M., Mendelsohn, N. (2004). *XML Schema Part 1: Structures*, second edition. W3C. http://www.w3.org/TR/xmlschema-1/.

UML (2007). *Unified Modeling Language*. Object Management Group. http://www.uml.org/.

XML Spy (2014). *Altova*. http://www.altova.com.

J. Klímek received his PhD. degree in Computer Science in September 2013 from the Charles University in Prague, Czech Republic, where he currently works at the Department of Software Engineering as a researcher. In addition, he works at the Faculty of Information Technology, Czech Technical University in Prague as an assistant professor at the Department of Software Engineering. His research areas involve Linked Data, XML data design, integration and evolution. He has published 23 refereed conference papers (2 of them Best Papers), 4 journal papers and 1 book chapter. He is a co-organizer of 1 local workshop and a PC member of an international conference. Jakub is a founding member of the OpenData.cz initiative.

J. Malý received his PhD degree in Computer Science in September 2013 from the Charles University in Prague, Czech Republic. His research areas involve conceptual modeling of XML data and evolution of XML applications, integrity constraints in models and Object Constraint Language. He has published 4 journal and 11 conference papers.

M. Nečaský received his PhD degree in Computer Science in 2008 from the Charles University in Prague, Czech Republic, where he currently works in the Department of Software Engineering as an assistant professor. His research areas involve XML data design and integration, Linked Data and Semantic Web. He has published more than 50 refereed conference papers and journal articles. Martin is a founding member of the OpenData.cz initiative.

I. Holubová received her PhD degree in Computer Science in 2007 from the Charles University in Prague, Czech Republic. Currently she is an associate professor at the Department of Software Engineering of the Charles University and an external member of the Department of Computer Science and Engineering of the Czech Technical University. She has published more than 60 publications in the area of XML data management and Web engineering, 4x she gained the Best Paper Award. She is a PC member or reviewer of 15 international events and co-organizer of 4 international workshops.

eXolution: metodika XML schemoms projektuoti ir modifikuoti panaudojant koncepcinį modeliavimą

Jakub KLÍMEK, Jakub MALÝ, Martin NEČASKÝ, Irena HOLUBOVÁ

Pastaruoju metu XML kalba tapo svarbiausia duomenų vaizdavimo kalba. Todėl su šia kalba siejamų technologijų ir tas technologijas naudojančių dalykinių programų skaičius sparčiai auga. Tačiau tik nedidelis procentas tokių programų yra statinės ir, jas įdiegus, nebėra keičiamos. Dauguma jų yra keičiamos, įgyvendinant naujus vartotojų keliamus reikalavimus arba dėl to, kad pakito jų vykdymo aplinka. Straipsnis aprašo programinį karkasą ir metodiką, skirtus tokiose programose naudojamų XML schemų evoliucijai ir jose padarytų pakeitimų pasekmių sklaidai per visą programą valdyti. Jis taip pat aprašo siūlomo programinio karkaso, pavadinto eXolution, eksperimentinę realizaciją, sukurtą per kelerius pastaruosius metus autorių ir jų bendradarbių. Straipsnis supažindina skaitytojus su plačia XML schemų evoliucijos tyrimų sritimi ir specifiniais autorių siūlomos metodikos bei ją palaikančio programinio karkaso realizacijos ypatumais.