XML Schema Integration with Reusable Schema Parts*

Jakub Klímek, Jakub Malý, and Martin Nečaský

XML Research Group, Department of Software Engineering
Faculty of Mathematics and Physics, Charles University in Prague
Malostranské náměstí 25, 118 00 Praha 1, The Czech Republic
{klimek, maly, necasky}@ksi.mff.cuni.cz

Abstract. Modern information systems may exploit numerous XML formats for communication. Each message may have its own XML format for data representation which causes problems with integration and evolution of their schemas. Manual integration and management of evolution of the XML formats may be very hard. We tackled this problem in our previous work, however, for simplicity reasons, we omitted the possibility of exploiting reusable schema parts. In this paper, we complement our previous work with additional methods for schema integration which exploit reusable schema parts that quite often appear in XML schemas. This further helps a domain expert to get a precise mapping to a conceptual diagram, which then integrates the XML formats and facilitates their evolution - a change that is made once in the conceptual diagram is propagated to the XML formats.

Keywords: XML schema, conceptual modeling, reverse-engineering, integration

1 Introduction

Today, XML is a standard for communication in various information systems like web services, etc. A web service provides an interface composed of several operations. The structure of incoming and outgoing messages is described in a form of XML schemas. If the XML schemas of communicating web services differ, the problem of their integration comes to the scene. When the XML schemas are integrated, another problem arises. Since the business evolves in time the XML schemas need to be adapted too.

We aim at the problem of integration of XML schemas by mapping them to a common conceptual schema. In our previous work [5, 10, 4], we have introduced a framework for XML schema integration and evolution. It supposes a set of XML schemas that are conceptually related to the same problem domain. As a problem domain, we can consider, e.g., purchasing products. Sample XML schemas may be XML schemas for purchase orders, product catalogue, customer detail, etc. The central part of the framework is a conceptual schema of the problem domain. Each XML schema is then mapped to the conceptual schema. In other words, the conceptual diagram integrates the XML schemas. We then exploit the mappings to evolve the XML schemas when a

* This work was supported in part by the Czech Science Foundation (GAČR), grant number P202/11/P455 and in part by the grant SVV-2011-263312.

V. Snášel, J. Pokorný, K. Richta (Eds.): Dateso 2011, pp. 13–24, ISBN 978-80-248-2391-1.

change occurs. Simply speaking, the change is made only once at the conceptual level and then propagated to the affected XML schemas. It is also possible to exploit the mappings to derive interfaces of semantic web services described in SAWSDL as we show in [11, 8]. In [7, 4] we have introduced a method of XML schema integration, which will be briefly introduced later.

Contributions In practice, a conceptual diagram and XML schemas exist separately, i.e. there are no mappings between both levels. This disallows to exploit the integration and evolution capabilities of our framework. In our work [9], we have introduced a method for deriving required XML schemas from the conceptual diagram and in [7] and [4] we have described a reversed method for mapping of an existing XML schema to the conceptual diagram. In this paper, we extend this method by utilizing inheritance constructs that often appear in XML schemas and that are supported by our conceptual model to get even better results and more comfortable way of integrating them.

Our aim is not to develop new methods for measuring schema similarities. These methods have been already intensively studied in the literature. Instead, we exploit the existing ones and combine them together. For this, we provide an algorithm skeleton that can be supplemented by various similarity methods. An important contribution of the method, not considered by existing similarity methods, is an active participation of a domain expert. This is necessary, since we need to achieve exact mapping.

Outline The rest of the paper is organized as follows. In Section 2, we present related work. In Section 3, we briefly present a simplified version of our conceptual model for XML. Section 4 briefly describes the algorithm from [7, 4] which assists a domain expert during mapping discovery and we enhance it with methods for dealing with inheritance. In Section 5, we evaluate the presented approach. Finally, Section 6 concludes.

2 Related work

Recent literature (surveyed in [6]) has been focused on a discovery of mappings of XML formats to a common model. We can identify several motivations. XML schemas are hardly readable and a friendly graphical notation is necessary. This motivation has appeared in [2][3] or [15]. A survey of these approaches can be found in [17]. They introduce an algorithm for automatic conversion of a given XML schema to a UML class diagram. The result exactly corresponds to the given XML schema. However, these approaches can not be applied in our case – we need to map an XML schema to an existing conceptual diagram. There are also approaches aimed at an integration of a set of XML format into a common XML format. These works include, e.g. the DIXSE framework [13] or Xyleme project [12]. Approaches that convert or map XML formats to ontologies are DTD2OWL [14], which presents a simple method of automatic translation of an XML format with an XML schema expressed in DTD into an ontology. More advanced methods are presented in [1] and [16]. They both introduce an algorithm that automatically maps an XML format to an ontology. This is close to our approach since a conceptual diagram can be understood as an ontology. In both cases, the domain expert can edit the discovered mappings but is not involved in the discovery process directly. For a more detailed description of related work see [7].

3 Our Conceptual Model

In this section, we will introduce our conceptual model for XML. We follow the Model-Driven Architecture (MDA) principle which is based on modeling data at several levels of abstraction. The most abstract level contains a conceptual schema of the problem domain. The language applied to express the conceptual schema is called *platform-independent model (PIM)*. The level below is the *platform-specific level* which specifies how the whole or a part of the PIM schema is represented in a particular platform. In our case, the platform is XML.

3.1 Platform-Independent Model

A PIM schema is based on UML class diagrams and models real-world concepts and relationships between them. It contains three types of components: classes, attributes and associations.

Definition 1. *Let $\mathcal{L}$ be a set of string labels and $\mathcal{D}$ be a set of datatypes. A* schema in the platform independent model (PIM schema) *is a 9-tuple $\mathcal{S} = (\mathcal{S}_c, \mathcal{S}_a, \mathcal{S}_r, \mathcal{S}_e, name, type, class, participant, card)$, where:*

- *$\mathcal{S}_c$ and $\mathcal{S}_a$ are sets of* classes *and* attributes *in $\mathcal{S}$, respectively.*
- *$\mathcal{S}_r$ is a set of* binary associations *in $\mathcal{S}$. $\mathcal{S}_e$ is a set of* association ends *in $\mathcal{S}$. A binary association is a set $R = \{E_1, E_2\}$, where E_1, $E_2 \in \mathcal{S}_e$ and $E_1 \neq E_2$. For any two associations R_1, $R_2 \in \mathcal{S}_r$ it must hold that $R_1 \cap R_2 \neq \emptyset \Rightarrow R_1 = R_2$. In other words, no two associations share the same end.*
- *$name : \mathcal{S}_c \cup \mathcal{S}_a \to \mathcal{L}$ resp. $name : \mathcal{S}_r \to \mathcal{L} \cup \{\lambda\}$ assigns a* name *to each class, attribute and association. $name(R) = \lambda$ means that $R \in \mathcal{S}_r$ does not have a name.*
- *$type : \mathcal{S}_a \to \mathcal{D}$ assigns a data type to each attribute.*
- *$class : \mathcal{S}_a \to \mathcal{S}_c$ assigns a class to each attribute. For $A \in \mathcal{S}_a$, we will say that A* is an attribute *of $class(A)$ or A* belongs to *$class(A)$.*
- *$participant : \mathcal{S}_e \to \mathcal{S}_c$ assigns a class to each association end. For $R = \{E_1, E_2\} \in \mathcal{S}_r$, we will say that $participant(E_1)$ and $participant(E_2)$* are participants of *R or that they* are connected by *R.*
- *$card : (\mathcal{S}_a \cup \mathcal{S}_e) \to \mathcal{C}$ assigns a cardinality to each attribute and association end.*

The members of $\mathcal{S}_c$, $\mathcal{S}_a$, and $\mathcal{S}_r$ are called components *of $\mathcal{S}$.*

We display PIM schemas as UML class diagrams. A class is displayed as a box with its name at the top and attributes at the bottom. An attribute is displayed as a pair comprising the attribute name and cardinality. The data type is omitted to make the diagram easy to read. An association is displayed as a line connecting participating classes with the association name and cardinalities.

For a given association $R = (E_1, E_2)$, we will often use notation (C_1, C_2) as an equivalent of $(participant(E_1), participant(E_2)$ if there are no more associations connecting C_1 and C_2. We will also need a construct called a PIM path.

Definition 2. *A* PIM path *P is an ordered sequence $\langle R_1, \ldots, R_n \rangle$ of associations from $\mathcal{S}_r$, where $(\forall i \in \{1, n\})((R_i) = (C_{i-1}, C_i))$. C_0 and C_n are called* start *and* end *of P. Functions $start$ and end return for P the start and end of P, respectively.*

An example of a PIM path in our sample PIM depicted in Figure 1(a) is $Path = \langle (Purchase, Item), (Item, Product), (Product, Supply) \rangle$. *Purchase* and *Supply* are start and end of the PIM path, respectively.

3.2 Platform–Specific Model

The *platform-specific model (PSM)* enables to specify how a part of the reality is represented in a particular XML schema in a UML-style way. We introduce it formally in Definition 3. We view a PSM schema in two perspectives. From the *grammatical perspective*, it models XML elements and attributes. From the *conceptual perspective*, it delimits the represented part of the reality. Its advantage is clear – the designer works in a UML-style way which is more comfortable then editing the XML schema.

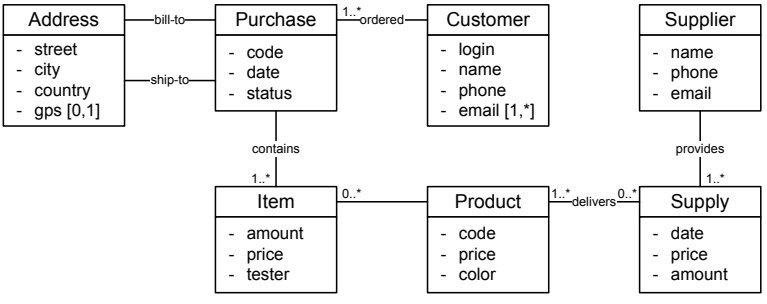
Definition 3. *Let $\mathcal{L}$ be a set of string labels and $\mathcal{D}$ be a set of datatypes. A* PSM schema *is a* 16*-tuple $\mathcal{S}' = (\mathcal{S}'_c, \mathcal{S}'_a, \mathcal{S}'_r, \mathcal{S}'_e, \mathcal{S}'_m, C'_{\mathcal{S}'}, name', type', class', xform', participant', card', cmtype', attributes', content', repr')$, where*

- *$\mathcal{S}'_c$, $\mathcal{S}'_a$, and $\mathcal{S}'_m$ are sets of* classes, attributes, *and* content models *in $\mathcal{S}'$, respectively.*
- *$\mathcal{S}'_r$ is a set of* directed binary associations *in $\mathcal{S}'$. $\mathcal{S}'_e$ is a set of* association ends *in $\mathcal{S}'$. A directed binary association is a pair $R' = (E'_1, E'_2)$, where E'_1, $E'_2 \in \mathcal{S}'_e$ and $E'_1 \neq E'_2$. For any two associations R'_1, $R'_2 \in \mathcal{S}'_r$ it must hold that $R'_1 \cap R'_2 \neq \emptyset \Rightarrow R'_1 = R'_2$.*
- *$C'_{\mathcal{S}'} \in \mathcal{S}'_c$ is a class called* schema class *of $\mathcal{S}'$.*
- *$name' : \mathcal{S}'_c \cup \mathcal{S}'_a \to \mathcal{L}$ resp. $name' : \mathcal{S}'_r \to \mathcal{L} \cup \{\lambda\}$ assigns a* name *to each class, attribute and association.*
- *$type' : \mathcal{S}'_a \to \mathcal{D}$ assigns a data type to each attribute.*
- *$class' : \mathcal{S}'_a \to \mathcal{S}'_c$ assigns a class to each attribute. For $A' \in \mathcal{S}'_a$, we will say that A'* is an attribute *of $class'(A')$ or A'* belongs to *$class'(A')$.*
- *$participant' : \mathcal{S}'_e \to \mathcal{S}'_c \cup \mathcal{S}'_m$ assigns a class or content model to each association end. For $R' = (E'_1, E'_2)$, where $X'_1 = participant'(E'_1)$ and $X'_2 = participant'(E'_2)$, we call X'_1 and X'_2* parent *and* child *of R', respectively. We will also sometimes call both X'_1 and X'_2* participants *of R' and say that X'_1 is the* parent *of X'_2 and X'_2 is a* child *of X'_1, denoted $parent'(R')$ and $child'(R')$, respectively.*
- *$xform' : \mathcal{S}'_a \to \{e, a\}$ assigns an* XML form *to each attribute. It specifies the XML representation of an attribute using an XML element declaration with a simple content or an XML attribute declaration, respectively.*
- *$card' : \mathcal{S}'_a \cup \mathcal{S}'_e \to \mathcal{C}$ assigns a* cardinality *to each attribute and association end.*
- *$cmtype' : \mathcal{S}'_m \to \{$`sequence`, `choice`, `set`$\}$ assigns a* content model type *to each content model. We distinguish 3 types: sequence, choice and set, respectively.*
- *$attributes' : \mathcal{S}'_c \to 2^{(\mathcal{S}'_a)}$ assigns an ordered sequence of distinct attributes to each class C'. It must hold that $A' \in attributes'(C') \Leftrightarrow C' = class'(A')$.*
- *$content' : \mathcal{S}'_c \cup \mathcal{S}'_m \to 2^{(\mathcal{S}'_r)}$ assigns an ordered sequence of distinct associations to each class or content model X'. It must hold that $R' \in content'(X') \Leftrightarrow X'$ is the parent of R'.*

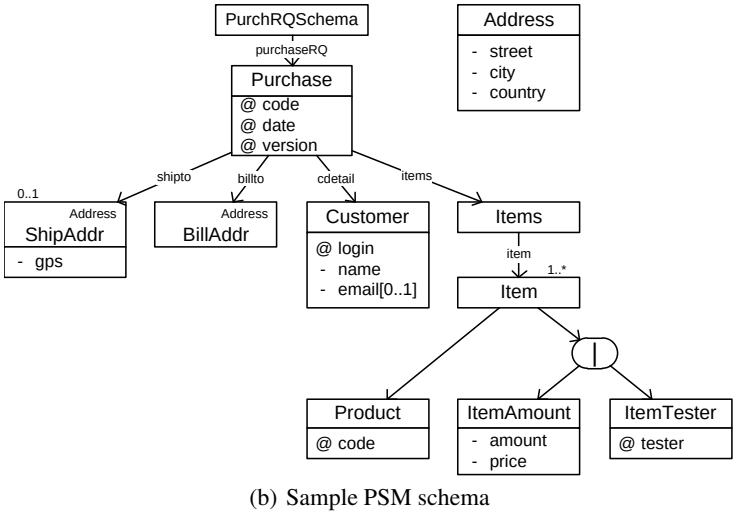
– $repr' : \mathcal{S}'_c \setminus \{C'_{\mathcal{S}'}\} \rightarrow \mathcal{S}'_c \setminus \{C'_{\mathcal{S}'}\}$ *assigns a class* $\overline{C'}$ *to another class* C'. C' *is called* structural representative *of* $\overline{C'}$. *It must hold that* $C' \notin repr'(\overline{C'})$. *Neither* C', *nor* $\overline{C'}$ *can be the schema class.*

The graph $(\mathcal{S}'_c \cup \mathcal{S}'_m, \mathcal{S}'_r)$ *with classes and content models as nodes and associations as directed edges must be a directed forest with one of its trees rooted in the schema class* $C'_{\mathcal{S}'}$. *Members of* $\mathcal{S}'_c$, $\mathcal{S}'_a$, $\mathcal{S}'_r$, *and* $\mathcal{S}'_m$ *are called* components *of* $\mathcal{S}'$.

A sample PSM schema is depicted in Figure 1(b). As can be seen from the definition, PSM introduces similar constructs to PIM: classes, attributes and associations.



(a) Sample PIM schema



(b) Sample PSM schema

Fig. 1. Sample PIM and PSM schemas

The PSM-specific constructs have precisely defined semantics. Briefly, a class models a complex content. The complex content is specified by the attributes of the class and associations in its content (their ordering is given by functions $attributes'$ and $content'$). An attribute models an XML element declaration with a simple content or

XML attribute declaration depending on its XML form (function $xform'$). An association models an XML element declaration with a complex content if it has a name. Otherwise, it models only that the complex content modeled by its child is nested in the complex content modeled by its parent. If a class C' is a structural representative of another class $repr'(C')$, the complex content modeled by C' extends the complex content modeled by $repr'(C')$. This is exactly our definition of a *reusable schema part* as multiple PSM classes can be structural representatives of another target PSM class, meaning that all of them reuse the definition of the target PSM class.

The PSM schema represents a part of a PIM schema. A class, attribute or association in the PSM schema may be mapped to a class, attribute or association in the PIM schema. In other words, there is a mapping which specifies the semantics of classes, attributes and associations of the PSM schema in terms of the PIM schema. The mapping must meet certain conditions to ensure consistency between PIM schemas and the specified semantics of the PSM schema. This mapping is called *interpretation of the PSM schema against the PIM schema*.

Definition 4. *Let $R = \{E_1, E_2\} \in \mathcal{S}_r$ be an association. An* ordered image *of R is a pair $R^{E_1} = (E_1, E_2)$ (or $R^{E_2} = (E_1, E_2)$).*

We will use $\overrightarrow{\mathcal{S}_r}$ to denote the set of all ordered images of associations of $\mathcal{S}'$, i.e. $\overrightarrow{\mathcal{S}_r} = \bigcup_{R \in \mathcal{S}'_r} \{R^{E_1}, R^{E_2}\}$. We need these definitions to be able to distinguish direction of PIM association, which is normally not needed in PIM.

Definition 5. *An* interpretation *of a PSM schema $\mathcal{S}'$ against a PIM schema $\mathcal{S}$ is a partial function $I : (\mathcal{S}'_c \cup \mathcal{S}'_a \cup \mathcal{S}'_r) \to (\mathcal{S}_c \cup \mathcal{S}_a \cup \overrightarrow{\mathcal{S}_r})$ which maps a class, attribute or association from $\mathcal{S}'$ to a class, attribute or ordered image of an association from $\mathcal{S}$, respectively. For $X' \in (\mathcal{S}'_c \cup \mathcal{S}'_a \cup \mathcal{S}'_r)$, we call $I(X')$* interpretation of X'. *$I(X') = \lambda$ denotes that I is not defined for X'. In that case, we will also say that X'* does not have an interpretation.

Let a function $context'_I : \mathcal{S}'_c \cup \mathcal{S}'_a \cup \mathcal{S}'_r \cup \mathcal{S}'_m \to \mathcal{S}'_c$ return for a given component X' of $\mathcal{S}'$ the closest ancestor class to X' on $path'(X')$ so that $I(context'_I(X')) \neq \lambda$. The following conditions must be satisfied:

$$I(C'_{\mathcal{S}'}) = \lambda \tag{1}$$

$$(\forall C' \in \mathcal{S}'_c \text{ s.t. } repr'(C') \neq \lambda)(I(C') = I(repr'(C'))) \tag{2}$$

$$(\forall A' \in \mathcal{S}'_a \text{ s.t. } I(A') \neq \lambda)(class(I(A')) = I(context'_I(A'))) \tag{3}$$

$$(\forall R' \in \mathcal{S}'_r \text{ s.t. } I(child'(R')) = \lambda)(I(R') = \lambda) \tag{4}$$

$$(\forall R' \in \mathcal{S}'_r \text{ s.t. } I(child'(R')) \neq \lambda)(I(R') = (I(context'_I(R')), I(child'(R')))) \tag{5}$$

Each PSM class, attribute or association can have an *interpretation* against a component of the PIM schema. This mapping means that in the PSM schema the particular PSM component models the concept represented by the target PIM component in the PIM schema.

Note that in the context of mapping of a PSM schema to a PIM schema (interpretation construction), the content models present in a PSM schema are irrelevant as they do not influence the semantics of PSM classes, attributes nor associations.

4 Algorithm

In this section we will enhance our interpretation reconstruction algorithm first introduced in [7] and extended to a framework in [4] so that it takes into account for reusable schema parts. These are represented in our conceptual model as *structural representants*. Because of lack of space in this paper, we will omit some details of the basic algorithm, which can be found in [7, 4].

The algorithm builds an interpretation I of a PSM schema against a PIM schema. I must be correct, it must fulfil Definition 5. Moreover, it must be correct in the conceptual sense, i.e. a PSM component and its PIM interpretation must conceptually correspond to the same real-world concept. We ensure the formal correctness. The conceptual correctness is ensured by a domain expert.

4.1 Overview

The basic algorithm works in three phases. Firstly, it measures initial similarities between PSM and PIM attributes and classes. Secondly, it creates an *initial interpretation* of PSM classes, whose initial similarity to some PIM class is higher than a given threshold. Becasue this is done automatically, there is a possibility that this initial interpretation is not correct. Therefore, it has to be verified by a domain expert. Nevertheless, the initial interpretation usually helps to avoid confirming lots of obvious mapping matches because the domain expert just needs to confirm a list of pre-mapped classes (or uncheck the incorrect ones). The confirmed initial interpretation now becomes a final interpretation and the algorithm moves to its third phase. It builds interpretation of the unmapped PSM classes with an assistance of a domain expert.

We will suppose a PSM schema $\mathcal{S}'$ and a PIM schema $\mathcal{S}$ on the input. The output of the algorithm is an interpretation I of $\mathcal{S}'$ against $\mathcal{S}$. We will enhance parts of the algorithm where the knowledge of reusable schema parts (structural representants) can help. But first, let us motivate a definiton. Let C' be a structural representant of C''. Due to condition 2 of Definition 5, the following must hold: $I(C') = I(C'')$. This means that both C' and C'' need to have the same interpretation in the PIM schema (or both must remain uninterpreted). This also means (from condition 3 of Definition 5 and from the definition of $context'(C')$), that PSM attributes of C' and C'' can only have attributes of the same PIM class as interpretation. Intuitively, C' and C'' represent the same concept in the PSM schema and we can suppose that their names also refer to the same concept. Note that the same goes for every PSM class C''', that would be a structural representant of C'. This justifies the follwing definition.

Definition 6. *Let a function $ss' : \mathcal{S}'_c \rightarrow 2^{(\mathcal{S}'_c)}$ return for each PSM class C' a set of PSM classes, which are (transitively) related to C' by the* structural representative *($repr'$) relation.*

For example, let C'_1, C'_2, C'_3 and C'_4 be PSM classes. Let $repr'(C'_1) = \lambda$, $repr'(C'_2) = C'_1$, $repr'(C'_3) = C'_1$ and $repr'(C'_4) = \lambda$. Then $ss'(C'_1) = ss'(C'_2) = ss'(C'_3) = \{C'_1, C'_2, C'_3\}$ and $ss'(C'_4) = \emptyset$.

4.2 Measuring Initial Similarity

Attributes. Firstly, the algorithm measures a similarity for each pair of one PIM and one PSM attribute. This is based on their names and datatypes. This phase is not affected by the structural representants and we can skip the detailed description. Suffice to say that results of initial attribute similarity are used in function $S^{init-attrs}(C', C)$ below, which gives us similarity of a PSM class and a PIM class based on their attributes.
Classes. Let $(C', C) \in \mathcal{C}' \times \mathcal{C}$. The similarity between C' and C is customizable, in this paper it is a weighted sum

$$S^{init-class}(C', C) = w^{init-class} * S^{init-attrs}(C', C) + (1 - w^{init-class}) * max\{S^{str}(name'(C'), name(C)), S^{str}(xml'(C'), name(C))\}$$

where $w^{init-class} \in (0, 1)$ is a weighting factor and $xml'(C')$ is a name of the parent association of C' if any exists. $S^{init-attrs}(C', C)$ is defined as $S^{init-attrs}(C', C) = \sum_{A' \in attributes'(C')} max_{A \in attributes(C)} (S^{init-attr}(A', A))$, i.e. it finds for each PSM attribute $A' \in attributes'(C')$ the most similar PIM attribute A of C and summarizes these similarities.

This is the first place where we can exploit structural representants. For a PSM class C', we can take attibutes of every $C'' \in ss'(C')$, because if those classes have an interpretation, it is the same PIM class for all of them (and similarly for the attributes). Therefore, we define function $attrs_{sr} : \mathcal{S}'_c \rightarrow 2^{(\mathcal{S}'_a)} = \cup_{C'_i \in ss'(C')} attributes'(C'_i)$ and we can redefine:

$$S^{init-attrs}(C', C) = \sum_{A' \in attrs_{sr}(C')} max_{A \in attributes(C)} (S^{init-attr}(A', A))$$

4.3 Initial interpretation

The initial class interpretations are set according to the initial class similarities pre-computed in the previous step. It is a simple procedure that takes the most similar pairs of PSM and PIM classes (with similarity greater than a given threshold) and sets these pairs as initial interpretations. Here is another place were we exploit structural representants. Because of the fact that all PSM classes of $ss'(C')$ need to have the same interpretaion (or none at all), when we initially interpret one of them, we can as well initially interpret all of them and the interpretation will be the same PIM class. And, of course, due to the possibility that this interpretation is incorrect, we can provide the user with the comfort of accepting/rejecting the whole group at once. If the domain expert chose to consider structural representatives in both the attribute similarity and the name similarity, this is an effect of the previous modification. The reason for this is that all of the classes from the group will have the same initial similarities, because when we computed the initial similarities for one class from the group, we included all the other classes as well. If, however, the domain expert chose to ignore structural representants at some stage, the similarities will be different and this adjustment may come in handy.

4.4 Final Interpretation

The third part of the algorithm iteratively traverses the PSM classes in $\mathcal{S}'_c$ in pre-order and helps the domain expert to build the final interpretation. Individual steps are shown

in Algorithm 1. For an actual PSM class $C' \in \mathcal{S}'_c$, the algorithm firstly constructs $I(C')$ (lines 2 - 6) and also sets the interpretation for all the PSM classes of $ss'(C')$ (lines 7 - 9). This is because all of them must have the same interpretation. Secondly, the algorithm constructs $I(A')$ for each $A' \in attributes(C')$ (lines 10 - 22). Finally, it constructs $I(R')$ for each $R' \in content(C')$ (lines 23 - 25). It can be shown that this algorithm runs in $O(N^3)$ where N is the number of PSM classes and in $O(n \times log(n))$ where n is the number of PIM classes.

Algorithm 1 Interpretation Construction Algorithm

1: **for all** $C' \in \mathcal{S}'_c$ in post-order **do**
2: **for all** $C \in \mathcal{S}_c$ **do**
3: $S^{class}(C', C) \leftarrow w^{class} * S^{init-class}(C', C) + (1 - w^{class}) * \frac{1}{S^{adj-class}(C', C)}$
4: **end for**
5: Offer the list of PIM classes sorted by S^{class} to the domain expert.
6: $I(C') \leftarrow C$ where $C \in \mathcal{S}_c$ is the PIM class selected by the domain expert.
7: **for all** $C'' \in ss'(C')$ **do**
8: $I(C'') \leftarrow C$ {here we set the interpretation for the whole group of PSM classes}
9: **end for**
10: **for all** $A' \in attributes(C')$ **do**
11: **for all** $A \in \mathcal{S}_a$ **do**
12: $S^{attr}(A', A) \leftarrow w^{attr} * S^{init-attr}(A', A) + (1 - w^{attr}) * \frac{1}{\mu(I(C'), class(A)) + 1}$
13: **end for**
14: Offer the list of PIM attributes sorted by S^{attr} to the domain expert.
15: $I(A') \leftarrow A$ where $A \in \mathcal{S}'_a$ is the PIM attribute depicted by the domain expert.
16: **if** $I(class'(A')) \neq class(A)$ **then**
17: Create PSM class $D' \in \mathcal{S}'_c$; $I(D') \leftarrow class(A)$
18: Put A' to $attributes'(D')$
19: Create PSM association $R' = (C', D') \in \mathcal{S}'_r$
20: Put R' at the beginning of $content'(C')$.
21: **end if**
22: **end for**
23: **for all** $R' \in content'(C')$ **do**
24: $I(R') \leftarrow P$ where P is the PIM path connecting $I(C')$ and $I(child'(R'))$ s.t. $\mu(I(C'), I(child'(R')))$ is minimal.
25: **end for**
26: **end for**

Class Interpretation To construct $I(C')$, the algorithm firstly computes $S^{class}(C', C)$ for each $C \in \mathcal{S}'_c$ at line 3. It is a weighted sum of two similarities. The former is the initial similarity $S^{init-class}(C', C)$. The other is a reversed class similarity adjustment $S^{adj-class}(C', C)$ which we discuss in a while. The algorithm then sorts the PIM classes by their similarity with C' and offers the sorted list to the domain expert at line 5. The expert selects a PIM class from the list and the algorithm sets $I(C')$ to this selected class at line 6.

Class similarity adjustment $S^{adj-class}(C', C)$ is computed on the base of the completed part of I, which includes confirmed initial interpretation. $S^{adj-class}(C', C)$ is a combination of distances between C and PIM classes D_i which are interpretations of the interpreted neighbors of C'. $\mu(C, D)$ is the distance between PIM classes C and D.

Note that Algorithm 1 is a skeleton which needs to be supplemented with methods for (1) measuring distances between PIM classes, (2) combining distances, and (3) selecting candidates for C' structural similarity adjustment. In this paper, we use basic methods to show that the general idea works. For measuring the distance between two PIM classes C and D, we use the length of the shortest PIM path connecting C and D. As the distance combination method, which results in the aimed $S^{adj-class}(C', C)$, we can also choose from various possibilities. In this paper, we use

$$S^{adj-class}(C', C) = (\sum_{i=1}^{n} \frac{\mu(C, I(D'_i))}{n}) + 1$$

where D'_1, ..., D'_n are the selected interpreted neighbors of C'. $S^{adj-class}(C', C)$ is the average of the lengths of the shortest PIM paths between C and each $I(D'_i)$.

Finally, we need to decide which mapped neighbors of C' will be selected to compute $S^{adj-class}(C', C)$. We can choose among children of C' or previous siblings of C', as these were already interpreted by the domain expert in this part of the algorithm. Because we have some PSM classes interpreted via the *initial interpretation*, we can use them as another candidates for structural similarity adjustment, if they are close enough. Therefore, we can also select interpreted following siblings, interpreted parent or interpreted ancestors as candidates for structural similarity adjustment. These options are described and experimented with in [4].

Here is another moment where we can exploit reusable schema parts in a form of structural representants. As we choose which interpreted neighbors of C' to use for the structural similarity adjustment, we can also work with the same type of interpreted neighbors of all classes of the group $ss'(C')$. The reasons are the same, because the interpretation of all classes of the group must be the same PIM class.

The rest of the algorithm remains unaffected by the structural representatives, so we describe it only briefly. For details, see [7, 4]. When all the PSM classes have been interpreted or the domain expert decided they should remain uninterpreted, a similar process is performed for PSM attributes of the classes. The possibilities of mapping a PSM attribute in this situation are limited due to the rules that the interpretation must adhere to (see Definition 5). Finally, PSM associations are interpreted with respect to the same rules.

5 Evaluation

In this section, we briefly evaluate the effect of structural representants on building interpretations of PSM classes. For more detailed experiments with the overall method see [7, 4]. We have implemented the introduced method in our tool XCase[1] which was primarily intended for designing XML schemas from a created PIM schema.

[1] `http://xcase.codeplex.com`

Let us suppose an actual PSM class C'. Let the domain expert set $I(C')$ to a PIM class C either when asked or when confirming the initial interpretation. We measure the precision of the algorithm from two points of view. Firstly, we measure the position of C in the list of PIM classes offered to the expert sorted by their S^{class}. We call this precision a *global precision* $\mathcal{P}_G$:

$$\mathcal{P}_G = ((\sum_{C' \in \mathcal{S}'_c} 1 - \frac{order(C) - 1}{n})/n') * 100$$

where n denotes the size of $\mathcal{S}_c$, n' denotes the size of $\mathcal{S}'_c$, and $order(C)$ denotes the order of C in the list. If there are more PIM classes with the same similarity to C', $order(C)$ is the order of the last one. $\mathcal{P}_G = 0$ (resp. 1) if for each PSM class C', the selected PIM class was the last (resp. first).

The global precision is not sufficient. When C is the first class, there can be other PIM classes before C which have their similarity to C' close to $S^{class}(C', C)$ and make it harder to distinguish whether C is or is not a good match for C'. We therefore propose another metric called *local precision* which measures the amount of PIM classes with their similarity to C' close to $S^{class}(C', I(C'))$. It is defined as

$$\mathcal{P}_L = ((\sum_{C' \in \mathcal{S}'_c} 1 - \frac{close(C) - 1}{n})/n') * 100$$

where $close(C)$ denotes the number of PIM classes with their similarity to C' close to $S^{class}(C', C)$. The term *close similarity* can be defined in various ways. In this paper, we say that y is *close* to x if $y \in (x - 0.1, x + 0.1)$.

Intuitively, the effect of using structural representants is a reduction of the number of mapping offers the domain expert needs to go through. This is because when an interpretation of a PSM class C' is constructed, it is automatically constructed for all PSM classes $ss'(C')$ and the domain expert no longer needs to create the interpretation for each one of them.

Additionally, the use of structural representants for class similarity computations may help with global and local precisions. This is, however, dependent on the texts present in the source PSM schema, its structure and selected methods of similarity measurements, which so far can not be determined automatically. Therefore, the experimental results are very complex and their description would not fit into this article.

6 Conclusion

In this paper, we studied the effect of exploiting reusable schema parts on techniques used for mapping of XML formats to a conceptual diagram. We briefly described our basic algorithm from [7, 4] which allows to exploit various similarity measurement methods. Then we introduced our enhancements that allow us to take advantage of the reusable schema parts, which are expressed as structural representants in our conceptual model. Finally, we have provided a biref evaluation of the proposed method.

References

1. R. dos Santos Mello and C. A. Heuser. A Bottom-Up Approach for Integration of XML Sources. In *Workshop on Information Integration on the Web*, pages 118–124, 2001.
2. J. Fong, S. K. Cheung, and H. Shiu. The XML Tree Model - toward an XML conceptual schema reversed from XML Schema Definition. *Data Knowl. Eng.*, 64(3):624–661, 2008.
3. M. R. Jensen, T. H. Møller, and T. B. Pedersen. Converting XML Data to UML Diagrams For Conceptual Data Integration. In *In Proceedings of DIWeb01*, 2001.
4. J. Klímek, I. Mlýnková, and M. Nečaský. A Framework for XML Schema Integration via Conceptual Model. In *Advances in Web, Intelligent, Cloud, and Mobile Systems Engineering - WISE 2010 Symposium and Workshops*. Springer, 2011.
5. J. Klímek and M. Nečaský. Integration and Evolution of XML Data via Common Data Model. In *Proceedings of the 2010 EDBT/ICDT Workshops, Lausanne, Switzerland, March 22-26, 2010*, New York, NY, USA, 2010. ACM.
6. J. Klímek and M. Nečaský. Reverse-engineering of XML Schemas: A Survey. In J. Pokorný, V. Snásel, and K. Richta, editors, *DATESO*, volume 567 of *CEUR Workshop Proceedings*, pages 96–107. CEUR-WS.org, 2010.
7. J. Klímek and M. Nečaský. Semi-automatic Integration of Web Service Interfaces. In *IEEE International Conference on Web Services (ICWS 2010)*, pages 307–314. IEEE Computer Society, 2010.
8. J. Klímek and M. Nečaský. Generating Lowering and Lifting Schema Mappings for Semantic Web Services. In *25th IEEE International Conference on Advanced Information Networking and Applications Workshops, WAINA 2010, Biopolis, Singapore, 22-25 March 2011*. IEEE Computer Society, 2011.
9. M. Nečaský. *Conceptual Modeling for XML*, volume 99 of *Dissertations in Database and Information Systems Series*. IOS Press/AKA Verlag, January 2009.
10. M. Nečaský and I. Mlýnková. On Different Perspectives of XML Schema Evolution. In *FlexDBIST'09*, Linz, Austria, 2009. IEEE.
11. M. Nečaský and J. Pokorný. Designing Semantic Web Services Using Conceptual Model. In *Proceedings of SAC'08*, pages 2243–2247. ACM, 2008.
12. C. Reynaud, J.-P. Sirot, and D. Vodislav. Semantic integration of xml heterogeneous data sources. In *In Proceedings of IDEAS '01*, pages 199–208, Washington, DC, USA, 2001. IEEE Computer Society.
13. P. Rodríguez-Gianolli and J. Mylopoulos. A Semantic Approach to XML-based Data Integration. In *ER '01: Proceedings of the 20th International Conference on Conceptual Modeling*, pages 117–132, London, UK, 2001. Springer-Verlag.
14. P. T. T. Thuy, Y.-K. Lee, and S. Lee. DTD2OWL: Automatic Transforming XML Documents into OWL Ontology. In *In Proceedings of ICIS '09*, pages 125–131, New York, NY, USA, 2009. ACM.
15. Y. Weidong, G. Ning, and S. Baile. Reverse Engineering XML. *Computer and Computational Sciences, International Multi-Symposiums on*, 2:447–454, 2006.
16. L. Xiao, L. Zhang, G. Huang, and B. Shi. Automatic Mapping from XML Documents to Ontologies. In *CIT '04: Proceedings of the The Fourth International Conference on Computer and Information Technology*, pages 321–325, Washington, DC, USA, 2004. IEEE Computer Society.
17. A. Yu and R. Steele. An Overview of Research on Reverse Engineering XML Schemas into UML Diagrams. In *ICITA (2)*, pages 772–777. IEEE Computer Society, 2005.