Similarity of XML Schema Definitions

Irena Mlýnková
Department of Software Engineering, Charles University in Prague, Czech Republic
irena.mlynkova@mff.cuni.cz

ABSTRACT

In this paper we propose a technique for evaluating similarity of XML Schema fragments. Firstly, we define classes of structurally and semantically equivalent XSD constructs. Then we propose a similarity measure that is based on the idea of edit distance utilized to XSD constructs and enables one to involve various additional similarity aspects. In particular, we exploit the equivalence classes and semantic similarity of element/attribute names. Using experiments we show the behavior and advantages of the proposal.

Categories and Subject Descriptors

I.7.1 [**Document and Text Processing**]: Document and Text Editing – languages, document management

General Terms

Measurement, Algorithms

Keywords

XML Schema, similarity, equivalence of XSD constructs.

1. INTRODUCTION

The eXtensible Markup Language (XML) [2] has become a standard for data representation and, thus, it appears in most of areas of information technologies. A possible optimization of XML-based methods can be found in exploitation of similarity of XML data. In this paper we focus on similarity of XML schemas that can be viewed from two perspectives. We can deal with either *quantitative* or *qualitative* similarity. In the former case we are interested in the degree of difference of the schemas, in the latter one we also want to know how the schemas relate, e.g. which of the schemas is more general. In this paper we deal with quantitative measure that is the key aspect of *schema mapping* [4, 5], i.e. searching for (sub)schemas that describe the same reality.

In this area the key emphasis is currently put on the semantic similarity of element/attribute names reflecting the requirements of corresponding applications. And if the approaches consider schema structure, they usually analyze only simple aspects such as, e.g., leaf nodes or child nodes. In addition, most of the approaches deal with XML schemas expressed in simple DTD language [2].

In this paper we focus on similarity of XML schema fragments in XML Schema language [9, 1]. In particular, we cover key XML Schema constructs and we deal with their structural and semantic equivalence. We propose a similarity measure that is based on the idea of edit distance utilized to XSD[1] constructs and enables one to involve various additional similarity aspects. In particular, we exploit the equivalence classes of XML constructs and semantic similarity of element/attribute names. Using experiments we show the behavior and advantages of the proposed approach.

2. EQUIVALENCE OF XSD CONSTRUCTS

The XML Schema language contains plenty of "syntactic sugar", i.e. constructs that enable one to generate XSDs that have different structure but are *structurally equivalent.*

DEFINITION 1. *Let S_x and S_y be two XSD fragments. Let $I(S) = \{D$ s.t. D is an XML document fragment valid against $S\}$. Then S_x and S_y are* structurally equivalent*, $S_x \sim S_y$, if $I(S_x) = I(S_y)$.*

Consequently, having a set X of all XSD constructs, we can specify the quotient set $X/\sim$ of X by $\sim$ and respective equivalence classes – see Table 1.

As depicted in Figure 1, for instance classes C_{ST} specify that there is no difference if a simple type is defined locally or globally, whereas class C_{Seq} expresses the equivalence between an unordered sequence of elements $e_1, e_2, ..., e_l$ and a choice of its all possible ordered permutations.

Apart from XSD constructs that restrict the allowed structure of XML data, we can find also constructs that express various semantic constraints. They involve identity constrains and simple data types `ID` and `IDREF(S)`. (Note that `ID`, `IDREF(S)` can be expressed using `key` and `keyref`.) The idea of semantic similarity is based on the following observation: A `keyref` construct refers to a particular part of the XSD – e.g. having an XSD containing a list of books and a list of authors, each author can refer to his best book. And this situation described in a semantically equivalent manner occurs when the referenced fragment, i.e. the element describing the best book, is directly present within author element. Hence, these constructs enable one to generate XSDs that have different structure but are *semantically equivalent.*

[1]XML Schema Definition

DocEng'08, September 16–19, 2008, San Paulo, Brazil.

Table 1: XSD equivalence classes of $X/\sim$

Class	Constructs	Canonical representative
C_{ST}	globally defined simple type, locally defined simple type	locally defined simple type
C_{CT}	globally defined complex type, locally defined complex type	locally defined complex type
C_{El}	referenced element, locally defined element	locally defined element
C_{At}	referenced attribute, locally defined attribute, attribute referenced via an attribute group	locally defined attribute
C_{ElGr}	content model referenced via an element group, locally defined content model	locally defined content model
C_{Seq}	unordered sequence of elements $e_1, e_2, ..., e_l$, choice of all possible ordered sequences of $e_1, e_2, ..., e_l$	choice of all possible ordered sequences of e_1, e_2, ..., e_l
C_{CTDer}	derived complex type, newly defined complex type	newly defined complex type
C_{SubSk}	elements in a substitution group G, choice of elements in G	choice of elements in G
C_{Sub}	data types $M_1, M_2, ..., M_k$ derived from type M, choice of content models defined in $M_1, M_2, ..., M_k, M$	choice of content models defined in $M_1, M_2, ..., M_k, M$

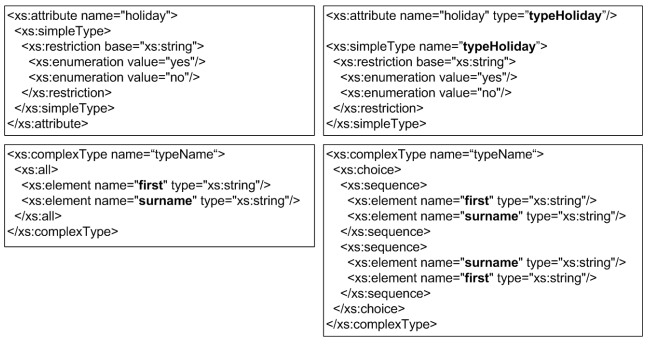

Figure 1: Examples for classes C_{ST} and C_{Seq}

DEFINITION 2. *Let S_x and S_y be two XSD fragments. Then S_x and S_y are* semantically equivalent*, $S_x \approx S_y$, if they abstract the same reality.*

Having a set X of all XSD constructs, we can specify the quotient set $X/\approx$ of X by $\approx$ and respective equivalence classes – see Table 2. Classes C'_{IdRef} and C'_{KeyRef} express the fact that both `IDREF(S)` and `keyref` constructs, i.e. references to schema fragments, are semantically equivalent to the situation when we directly copy the referenced schema fragments to the referencing positions. An example of the equivalent schemas is depicted in Figure 2.

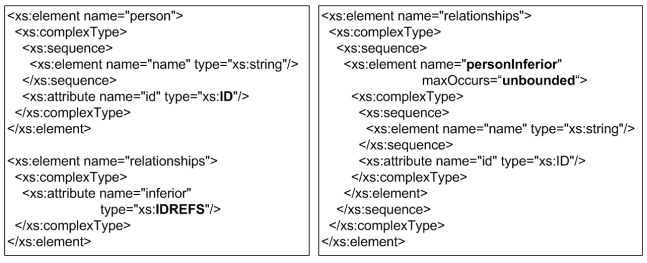

Figure 2: Example for class C'_{IdRef}

Since every `key`/`keyref` constraint must contain one reference (`selector`) to a set of elements and at least one reference (`field`) to their subelements (descendants) and/or attributes expressed in the following grammar [9, 1]:

```
Selector := PathS ('|' PathS)*
Field    := PathF ('|' PathF)*
PathS    := ('.//')? Step ('/' Step)*
PathF    := ('.//')? (Step '/')* (Step | '@' NameTest)
Step     := '.' | NameTest
NameTest := QName | '*' | NCName ':' '*'
```

the referenced fragments can be always easily copied to particular positions.

Naturally, each of the previously defined classes of $\sim$ or $\approx$ equivalence can be represented using a selected *canonical representative* as listed in Tables 1 and 2. (Not that each of the constructs not mentioned in Table 1 or 2 forms a single class $C_1, C_2, ..., C_n$ or $C'_1, C'_2, ..., C'_m$.)

3. SIMILARITY EVALUATION

The proposed algorithm is based mainly on the work presented in [7] which focuses on expressing similarity of XML documents using tree edit distance. The algorithm introduces two new edit operations *InsertTree* and *DeleteTree* which allow manipulating more complex structures than only a single node. But, repeated structures can be found in an XSD as well, if it contains *shared* fragments or *recursive* elements. In addition, we will concern both semantic equivalence of XSD fragments as well as semantic similarity of element/attribute names.

Firstly, the input XSDs S_x and S_y are parsed and their tree representations are constructed. Next, costs for tree inserting and tree deleting are computed. And in the final step we compute the resulting edit distance.

3.1 XSD Tree Construction

The key operation of our approach is tree representation of the given XSDs. However, since the structure of an XSD can be quite complex, we *normalize* and *simplify* it initially.

Firstly, we normalize the given XSDs using the equivalence classes. In the first step we exploit structural equivalence $\sim$ and we iteratively replace each non-canonical construct (naturally except for the root element) with the respective canonical representative until there can be found any. At the same time, for each XSD construct v of the schema we keep the set $v_{eq_\sim}$ of classes it originally belonged to. In the second step we exploit semantic equivalence $\approx$ and we again replace each non-canonical construct with its canonical representative and we construct sets $v_{eq_\approx}$. Now the resulting schema involves elements, attributes, operators `choice` and `sequence`, intervals of allowed occurrences, simple types and assertions. There are also no shared schema fragments.[2]

[2]We will omit the solution for recursive elements for the paper length, assuming that the input XSDs are non-recursive.

Table 2: XSD equivalence classes of $X/\approx$

Class	Constructs	Canonical representative
C'_{IdRef}	locally defined schema fragment, schema fragment referenced via `IDREF` attribute	locally defined schema fragment
C'_{KeyRef}	locally defined schema fragment, schema fragment referenced via `keyref` element	locally defined schema fragment

Next we simplify the remaining content models. For this purpose we use a subset of rules for DTD constructs from [8] as depicted in Figure 3.

I.a) $(e_1\|e_2)^* \rightarrow e_1^*, e_2^*$	II.a) $e_1^{+\,+} \rightarrow e_1^+$	II.b) $e_1^{*\,*} \rightarrow e_1^*$
I.b) $(e_1, e_2)^* \rightarrow e_1^*, e_2^*$	II.c) $e_1^{*}? \rightarrow e_1^*$	II.d) $e_1?^{*} \rightarrow e_1^*$
I.c) $(e_1, e_2)? \rightarrow e_1?, e_2?$	II.e) $e_1^{+\,*} \rightarrow e_1^*$	II.f) $e_1^{*\,+} \rightarrow e_1^*$
I.d) $(e_1, e_2)^+ \rightarrow e_1^+, e_2^+$	II.g) $e_1?^{+} \rightarrow e_1^*$	II.h) $e_1^{+}? \rightarrow e_1^*$
I.e) $(e_1\|e_2) \rightarrow e_1?, e_2?$	II.i) $e_1?? \rightarrow e_1?$	

Figure 3: Flattening and simplification rules

The rules enable one to convert all element definitions so that each cardinality constraint operator is connected to a single element. The second purpose is to avoid usage of `choice` construct. Note that some of the rules do not produce equivalent XML schemes and cause a kind of information loss. But this aspect is common for all existing XML schema similarity measures – it seems that the full generality of the regular expressions cannot be captured easily.

Now, having a normalized and simplified XSD, its tree representation is defined as follows:

DEFINITION 3. *An* XSD tree *is an ordered tree $T = (V, E)$, where*

1. *V is a set of nodes of the form $v = (v_{Type}, v_{Name}, v_{Cardinality}, v_{eq_\sim}, v_{eq_\approx})$, where v_{Type} is the type of a node (i.e. attribute, element or particular simple data type), v_{Name} is the name of an element or an attribute, $v_{Cardinality}$ is the interval $[v_{low}, v_{up}]$ of allowed occurrence of v, $v_{eq_\sim}$ is the set of classes of $\sim$ v belongs to and $v_{eq_\approx}$ is the set of classes of $\approx$ v belongs to,*
2. *$E \subseteq V \times V$ is a set of edges representing relationships between elements and their attributes or subelements.*

3.2 Tree Edit Operations

Having the above described tree representation of an XSD, we can now easily utilize the tree edit algorithm proposed in [7]. For a given tree T with a root node r of degree t and its first-level subtrees $T_1, T_2, ..., T_t$, the tree edit operations involve *substitution* of r with node r_{new} (operation *Substitute*), *inserting node x with degree 0* among $T_1, T_2, ..., T_t$ at position i (*Insert*), *deleting a leaf node T_i* (*Delete*), *inserting tree T_x* among $T_1, T_2, ..., T_t$ at position i (*InsertTree*) and *deleting tree T_i* (*DeleteTree*).

Transformation of a source tree T_x to a destination tree T_y can be done using a number of sequences of the operations. But, we can only deal with so-called *allowable* sequences, i.e. the relevant ones. For the purpose of our approach we only need to modify the original definition as follows:

DEFINITION 4. *A sequence of edit operations transforming a source tree T_x to a destination tree T_y is* allowable *if it satisfies the following two conditions:*

1. *A tree T may be inserted only if tree similar to T already occurs in T_x. A tree T may be deleted only if tree similar to T occurs in T_y.*
2. *A tree that has been inserted via the InsertTree operation may not subsequently have additional nodes inserted. A tree that has been deleted via the DeleteTree operation may not previously have had nodes deleted.*

While the original definition requires exactly the same trees, we relax the requirement only to similar ones.

3.3 Edit Distance

Inserting (deleting) a subtree T_i can be done with a single operation *InsertTree* (*DeleteTree*) or with a combination of *InsertTree* (*DeleteTree*) and *Insert* (*Delete*) operations. To find the optimal variant we adopt the algorithm from paper [7]; we only relax the conditions for exact matching of elements/attributes to their similarity. To determine the resulting edit distance, the algorithm uses pre-computed cost for inserting each T_i, $Cost_{Graft}(T_i)$ and deleting each tree T_i, $Cost_{Prune}(T_i)$. We omit the exact algorithm for space limitations referring an interested reader to the original paper and we describe only the similarity measure.

Similarity of Elements/Attributes.

Since the structural similarity is solved via the edit distance, we focus on semantic and syntactic similarity, cardinality-constraint similarity, similarity of equivalence classes and similarity of simple data types.

Semantic similarity of element/attribute names is a score that reflects the semantic relation between the meanings of two words. We exploit procedure described in [5] which determines ontology similarity between two words v_{Name} and v'_{Name} by comparing v_{Name} with synonyms of v'_{Name}.

Syntactic similarity of element/attribute names is determined by computing the edit distance between v_{Name} and v'_{Name}. For our purpose the classical Levenshtein algorithm [6] is used that determines the edit distance of two strings using inserting, deleting or replacing single characters.

Similarity of cardinality constraints is determined by similarity of intervals $v_{Cardinality} = [v_{low}, v_{up}]$ and $v'_{Cardinality} = [v'_{low}, v'_{up}]$. It is defined as follows:

$$
\begin{aligned}
&CardSim(v, v') \\
&= \quad 0 \quad ; (v_{up} < v'_{low}) \vee (v'_{up} < v_{low}) \\
&= \quad 1 \quad ; v_{up}, v'_{up} = \infty \wedge v_{low} = v'_{low} \\
&= \quad 0.9 \quad ; v_{up}, v'_{up} = \infty \wedge v_{low} \neq v'_{low} \\
&= \quad 0.6 \quad ; v_{up} = \infty \vee v'_{up} = \infty \\
&= \quad \frac{min(v_{up}, v'_{up}) - max(v_{low}, v'_{low})}{max(v_{up}, v'_{up}) - min(v_{low}, v'_{low})} \; ; \text{otherwise}
\end{aligned}
$$

Structural/semantic similarity of schema fragments rooted at v and v' is determined by the similarity of sets $v_{eq_\sim}$, $v'_{eq_\sim}$ and $v_{eq_\approx}$, $v'_{eq_\approx}$. It is defined as follows:

$$
\begin{aligned}
StrFragSim(v, v') \quad &= 1 \quad ; v_{eq_\sim}, v'_{eq_\sim} = \emptyset \\
&= \frac{|v_{eq_\sim} \cap v'_{eq_\sim}|}{|v_{eq_\sim} \cup v'_{eq_\sim}|} \; ; \text{otherwise} \\
SemFragSim(v, v') \quad &= 1 \quad ; v_{eq_\approx}, v'_{eq_\approx} = \emptyset \\
&= \frac{|v_{eq_\approx} \cap v'_{eq_\approx}|}{|v_{eq_\approx} \cup v'_{eq_\approx}|} \; ; \text{otherwise}
\end{aligned}
$$

And, finally, *similarity of data types* v_{Type} and v'_{Type} is specified by *type compatibility matrix* that determines similarity of distinct simple types. For instance, similarity of `string` and `normalizedString` is 0.9, whereas similarity of `string` and `positiveInteger` is 0.5. The table also involves similarity of restrictions of simple types, similarity between element and attribute nodes etc. (We omit the whole table for the paper length.)

The overall similarity, $Sim(v, v')$ of nodes v and v' is computed as follows:

$$\begin{aligned} & Sim(v, v') \\ = \;& Max(SemanticSim(v, v'), SyntacticSim(v, v')) \times \alpha_1 \\ + \;& CardSim(v, v') \times \alpha_2 \\ + \;& StrFragSim(v, v') \times \alpha_3 \\ + \;& SemFragSim(v, v') \times \alpha_4 \\ + \;& DataTypeSim(v, v') \times \alpha_5 \end{aligned}$$

where $\sum_{i=1}^{5} \alpha_i = 1$ and $\forall i : \alpha_i \geqslant 0$.

4. EXPERIMENTS

For the purpose of experimental evaluation of the proposal we have created three synthetic XSDs that are from various points of view more or less similar. XSD I and II differ within classes of $\sim$ equivalence, XSD III differs in more aspects, such as, e.g., simple types, allowed occurrences, globally/locally defined data types, element/attribute names, attributes vs elements with simple types etc. Results of all the tests are depicted in Table 3.

Table 3: Similarity of XSDs I, II and III

Test		I × II	II × III	III × I
A	$\alpha_3 = \alpha_4 = 0$	1.00	0.82	0.82
B	$\alpha_4 = 0$, $\alpha_3 \neq 0$	0.89	0.70	0.66
C	$\alpha_3 = 0$, $\alpha_4 \neq 0$	1.00	0.80	0.80
D	A without *SemanticSim*	1.00	0.33	0.33
E	B without *SemanticSim*	0.89	0.255	0.24

In Test A we set $\alpha_3 = \alpha_4 = 0$, i.e. we ignore the information on original constructs of XML Schema. As we can see, the similarity of XSD I and XSD II is 1.0, because they are represented using identical XSD trees. Similarity between XSD I vs XSD III and XSD II vs XSD III are for the same reason equivalent, though naturally lower.

If we set $\alpha_3 \neq 0$ in Test B (according to our experiments it should be > 0.2 to influence the algorithm), the resulting similarity is influenced by the difference between the used XML Schema constructs. As we can see, the results are more precise – the similarity of XSD I and II is naturally $\neq 1.0$ and similarity of XSD II and III is higher than of XSD III and I due to the respective higher structural similarity of constructs.

On the other hand, if we set $\alpha_4 \neq 0$ and $\alpha_3 = 0$ in Test C, i.e. we are interested in semantic similarity of schema fragments, the results have the same trend as results in Test A, because we again omit structural similarity of XML Schema constructs, but in this case the semantic similarity of schema fragments called `relationships` (occurring in XSD I and II) and `connections` (occurring in XSD III) specifies the result more precisely.

Finally, in Test D and E we focus on the most time consuming operation of the approach which determines the overall complexity of the algorithm – searching the thesaurus in operation *SemanticSim*. If we consider the first situation, i.e. when $\alpha_3 = \alpha_4 = 0$, it influences similarity with XSD III, whereas similarity of XSD I and II remains the same because the respective element/attribute names are the same. The results in case $\alpha_4 = 0$ are depicted in Test E. As we can see, the similarity of XSD I and II remains the same again, whereas the other values are much lower.

In general, the experiments show that various parameters of the similarity measure can highly influence the results. On the other hand, we cannot simply analyze all possible aspects, since some applications may not be interested, e.g., in semantic similarity of used element/attribute names or the "syntactic sugar"XML Schema involves. Consequently, a reasonable approach should enable one to exploit various aspects as well as temporarily omit the irrelevant ones.

5. FUTURE WORK

In our future work we will naturally focus on further improvements of our approach. We will deal with other edit operations (e.g. moving a node [3] or adding/deleting a non-leaf node), improvements of efficiency of supplemental algorithms, especially the semantic similarity, and on problems related to reasonable setting of involved weights. We will also deal with more elaborate experimental testing that will involve implementation of a simulator that would provide distinct XSDs.

6. ACKNOWLEDGEMENT

This work was supported by the National Programme of Research (Information Society Project 1ET100300419).

7. REFERENCES

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

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

[3] G. Cobena, S. Abiteboul, and A. Marian. Detecting Changes in XML Documents. In *ICDE'02*, pages 41–52. IEEE, 2002.

[4] H. H. Do and E. Rahm. COMA – A System for Flexible Combination of Schema Matching Approaches. In *VLDB'02*, pages 610–621. Morgan Kaufmann, 2002.

[5] M. L. Lee, L. H. Yang, W. Hsu, and X. Yang. XClust: Clustering XML Schemas for Effective Integration. In *CIKM'02*, pages 292–299. ACM Press, 2002.

[6] V. I. Levenshtein. Binary Codes Capable of Correcting Deletions, Insertions and Reversals. *Soviet Physics Doklady*, 10:707, 1966.

[7] A. Nierman and H. V. Jagadish. Evaluating Structural Similarity in XML Documents. In *WebDB'02*, pages 61–66, 2002.

[8] J. Shanmugasundaram, K. Tufte, C. Zhang, G. He, D. J. DeWitt, and J. F. Naughton. Relational Databases for Querying XML Documents: Limitations and Opportunities. In *VLDB'99*, pages 302–314. Morgan Kaufmann, 1999.

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