An XML-to-Relational User-Driven Mapping Strategy Based on Similarity and Adaptivity *

© Irena Mlynkova

Charles University
Faculty of Mathematics and Physics
Department of Software Engineering
Malostranske nam. 25
118 00 Prague 1, Czech Republic
irena.mlynkova@mff.cuni.cz

Abstract

As XML has become a standard for data representation, it is inevitable to propose and implement techniques for efficient managing of XML data. A natural alternative is to exploit features and functions of (object-)relational database systems, i.e. to rely on their long theoretical and practical history. The main concern of such techniques is the choice of an appropriate XML-to-relational mapping strategy.

In this paper we focus on enhancing of *user-driven* techniques which leave the mapping decisions in hands of users. We propose an algorithm which exploits the user-given annotations more deeply searching the user-specified "hints" in the rest of the schema and applies an adaptive method on the remaining schema fragments. We describe the algorithm theoretically, discussing the key ideas of the approach, chosen solutions, their reasons, and consequences. Finally, we overview the open issues related to implementation of the proposed algorithm and its experimental testing on real XML data.

1 Introduction

Without any doubt the eXtensible Markup Language (XML) [9] is currently de-facto a standard for data representation and manipulation. Its popularity is given by the fact that the basic W3C recommendations are well-defined, easy-to-learn, and at the same time still enough powerful. The popularity naturally invoked a boom of their efficient implementations based on various storage strategies from the traditional ones such as file systems to brand-new ones proposed particularly for XML structures, so-called *native* approaches or directly *native XML databases*.

Probably the most natural and practically used approach involves techniques which exploit features of (object-)relational database systems, though they are not as efficient as the native ones due to the key problem of structural differences between XML data and relations. The reason for the popularity is that relational databases are still regarded as universal and powerful data processing tools and their long theoretical and practical history can guarantee a reasonable level of reliability and efficiency. Contrary to native methods it is not necessary to start "from scratch" but we can rely on a mature and verified technology, i.e. properties that no native XML database can offer yet. On this account we believe that these methods and their possible improvements should be further enhanced.

The main concern of the database-based[1] XML techniques is the choice of an appropriate XML-to-relational mapping strategy, i.e. the way the given XML data are stored into relations. We can distinguish the following three types approaches [4] [15]:

- *fixed* mapping methods based on predefined set of mapping rules and appropriate heuristics (e.g. [11] [21]),
- *adaptive* methods which adapt the target database schema to the intended application (e.g. [12] [8]), and
- methods which leave the mapping decisions in hands of users (e.g. [3] [5]).

The first set of methods can be further divided [4] into *generic* and *schema-driven* ones depending on omitting or exploiting the existence of corresponding XML schema. However, both the types use a straightforward mapping strategy regardless the intended future usage. On the contrary, adaptive methods automatically adapt the database schema of a fixed method to the given additional information. The best known representatives, so-called *cost-driven* methods, usually search a space of possible XML-to-relational mappings and choose the one which conforms to the required application, specified using a sample set of XML data and XML queries, the most, i.e. where the provided queries over the given data can be evaluated most efficiently. Finally, the last

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

Proceedings of the Spring Young Researcher's Colloquium on Database and Information Systems, Moscow, Russia, 2007

[1] In the rest of the paper the term "database" represents an (object-) relational database.

mentioned type of methods can be also divided into two groups. We distinguish so-called *user-defined* and *user-driven* methods [15] which differentiate in the amount of necessary user interaction. In the former case a user is expected to define both the target database schema and the required mapping strategy, i.e. to do all the work "manually". In the latter case a default mapping strategy is defined but a user can specify local mapping changes from the predefined set of other allowed mapping strategies, usually using schema annotations. In other words, the user-driven approach solves the main disadvantage of the user-defined one – the requirement of a user skilled in two complex technologies who is, in addition, able to specify an optimal database schema for a particular application. Note that the user-driven techniques can be regarded as a type of adaptive methods too [15], since they also adapt the default target schema to additional information, in particular to user-specified requirements.

In this paper we focus on further enhancing of user-driven techniques, particularly on their (in our opinion) two main persisting disadvantages. The first one is the fact that the default mapping strategy is (to our knowledge) always a fixed one. It is quite a surprising finding since we know that the proposed systems are able to store schema fragments in various ways. From this point of view an adaptive enhancing of the fixed method seems to be quite natural and suitable. The second key shortcoming we are dealing with is weak exploitation of the user-given information. We believe that the schema annotations a user provides can not only be directly applied on particular schema fragments, but the information they carry can be further exploited. The main idea is quite simple – we regard the annotations as "hints" how a user wants to store particular XML patterns and we use this information twice again. Firstly, we search for similar patterns in the rest of the schema and store the found fragments in a similar way. And secondly, we exploit the information in the adaptive strategy for not annotated parts of the schema.

To sum up, the main contribution of this paper is a proposal of an algorithm which enhances classical user-driven strategies using the following two approaches:

- a deeper exploitation of the information carried in user-given schema annotations and
- an adaptive mapping strategy for not annotated parts of the schema.

We describe the proposed algorithm theoretically. We discuss the key ideas and problems, their possible solutions, reasons for our particular decisions, and their consequences. Finally, we overview the related problems and open issues of a particular implementation of the proposal.

The rest of the paper is structured as follows: Section 2 contains a motivation for focusing on user-given information. Section 3 overviews the existing related works in the area of user-driven methods, adaptive methods, and similarity of XML data. In the fourth section we describe and discuss the proposed algorithm in more detail and in the fifth section we sum up the corresponding implementation open issues. Finally, Section 6 provides conclusions and outlines our future work.

2 Motivation

The key concern of this paper is to exploit the user-given information as much as possible. We result from the idea of user-driven enhancing of the user-defined techniques, where a user is expected to help the mapping process, not to perform it. We want to go even farther. But first of all we discuss why user-given information is so important to deal with.

A simple demonstrative example can be a set of XML documents which contain various XHTML [1] fragments. A classical fixed schema-driven mapping strategy (e.g. [21] [14]) would decompose the fragments into a number of relations. Since we know that the standard XHTML DTD allows, e.g., complete subgraphs on up to 10 nodes, the reconstruction of such fragments would be a really expensive operation in terms of the number of join operations. But if we knew that the real complexity of such fragments is much simpler (and the analysis of real XML data shows that it is quite probable [17]), e.g. that each of the fragments can be described as a simple text with tags having the depth of 2 at most, we could choose a much simpler storage strategy including the extreme one – a CLOB column.

Another example can be the crucial feature of database storage strategies – updatability of the data. On one hand, we could know that the data will not be updated too much or at all but we need an effective query evaluation. On the other hand, there could be a strong demand for effective data updates, whereas the queries are of marginal importance. And there are of course cases which require effective processing of both. Naturally, the appropriate storage strategies differ strongly. In case of effective query processing a number of indices and numbering schemes can be exploited but at the cost of corresponding expensive updates. Effective updates, conversely, require the simplest information of mutual data relationships. And if both the aspects are required, it is unavoidable to compromise. And such decision can be again made correctly only if we have an appropriate information on the required future usage.

Last but not least, let us consider the question of data redundancy. Without any additional information the optimal storage strategy is so-called *4NF schema decomposition* into relations [5], where 4NF stands for the *fourth normal form*, which can be achieved, e.g., using the classical *Hybrid algorithm* [21], a representative of fixed mapping methods. The decomposition does not involve data redundancy or violation of any normal form, i.e. it results in a database schema with the lowest number of relations and null attributes. But, similarly to database design, there can be reasonable real-world cases when the data should not strictly follow the rules of normal forms and their moderation can lead to more effective query processing.

Both the cost-driven and user-driven methods are based on the idea of exploiting additional user-given information and they appropriately adapt the target database schema. In the former case it is extracted from a sample set of XML documents and/or XML queries which characterize the typical future usage, in the latter case it is specified by user-given annotations, i.e. the user directly specifies the required changes of the default mapping. But although there is a plenty of existing repre-

sentatives of the two approaches, there are still numerous weak points and open issues that should be improved and solved [15].

As mentioned above, our first improvement is searching for identical or similar fragments in the not annotated schema parts. This approach has two main advantages:

1. The user is not forced to annotate all schema fragments that have to be stored alternatively, but only those with different structure. Thus the system is not endangered of unintended omitting of annotating all similar cases.

2. The system can reveal structural similarities which are not evident "at first glance" and which could remain hidden to the user.

Thus the first main concern of our proposal is how to identify identical or similar fragments within the schema.

Our second enhancing focuses on the choice of the mapping strategy for schema fragments which were neither annotated by the user, nor identified as fragments similar to the annotated ones. In this case we combine the idea of cost-driven methods with the fact that a user-driven technique should support various storage strategies too. Hence our second concern is how to find the optimal mapping strategy for the remaining schema fragments and, in addition, with exploitation of the information we already have, i.e. the user-specified annotations, as much as possible.

3 Related Work

As we have mentioned, methods which involve a user in the mapping process can be divided into user-defined and user-driven. Probably due to simple implementation the former ones are supported in most commercial database systems [2]. On the other hand, the set of techniques of the latter type is surprisingly small. To our knowledge there are just two main representatives of the approach – so-called *Mapping Definition Framework (MDF)* [3] and *XCacheDB System* [5]. Both support inlining and outlining of an element / attribute, mapping an element / attribute to a CLOB column, renaming target tables / columns, and redefining column data types. The former approach furthermore supports the *Edge mapping* [11] strategy and enables to specify the required capturing of the structure of the whole schema. The latter one, in addition, allows a certain degree of redundancy.

In both the cases the mapping for not annotated parts is fixed and the annotations are applied just directly on the annotated schema fragments. The two ideas we want to use for their enhancing are adaptivity [15] and similarity [16].

3.1 Adaptive XML-to-Relational Mapping

Probably the first proposal of an adaptive cost-driven method can be found in [12]. It is based on the idea of storing well structured parts of XML documents into relations (using the 4NF decomposition) and semi-structured parts using an *XML data type*, which supports path queries and XML-aware full-text operations. The main concern of the method is to identify the structured and semi-structured parts. For this purpose a sample set of XML documents and XML queries is used.

The other existing cost-driven approaches [8] [24] [26] use a different strategy. They define a set of XML-to-XML transformations (e.g. inlining / outlining of an element / attribute, splitting / merging of a shared element[2], associativity, commutativity, etc.), a fixed XML-to-relational mapping, and a cost function which evaluates a relational schema against a given sample set of XML data and/or queries. Using a search algorithm a space of possible relational schemes is searched and the optimal one is selected. Since it can be proven that even a simple set of transformations causes the problem to be NP-hard, the corresponding search algorithms in fact search for suboptimal solutions and exploit, e.g., heuristics, terminal conditions, approximations, etc.

3.2 Similarity of XML Data

Exploitation of similarity of XML data can be found in various XML technologies, such as, e.g., document validation, query processing, data transformation, storage strategies based on clustering, data integration systems, dissemination-based applications, etc. Consequently, the number of existing works is enormous. We can search for similarity among XML documents, XML schemes, or between the two groups. Furthermore, we can distinguish several levels of similarity that can be taken into account during the search process – a structural level (i.e. considering only the structure of the given XML fragments), a semantic level (i.e. taking into account also the meaning of element / attribute names), a constraint level (i.e. taking into account also various text value constraints), etc.

In case of document similarity we distinguish techniques expressing the similarity of two documents D_1 and D_2 by measuring how difficult is to transform D_1 into D_2 (e.g. [19]) and techniques which specify a simple and reasonable representation of D_1 and D_2 that enables their efficient comparison and similarity evaluation (e.g. [25]). In case of similarity of document D and schema S there are also two types of strategies – techniques which measure the number of elements which appear in D but not in S and vice versa (e.g. [6]) and techniques which measure the closest distance between D and all documents valid against S (e.g. [18]). Finally, methods for measuring similarity of two XML schemes S_1 and S_2 exploit and combine various supplemental information and measures such as, e.g., predefined similarity rules, similarity of element / attribute names, equivalence of data types and structure, schema instances, thesauri, previous results, etc. (e.g. [13] [10] [22])

4 Proposed Algorithm

The general idea of fixed schema-driven XML-to-relational mapping methods is to decompose the given XML schema S into a set of *schema fragments* $F = \{f_1, f_2, ..., f_n\}$. Each $f_i \in F$ is then mapped to a corresponding relation r_i using a mapping strategy s_{frag}. The relationship between fragments is kept using a mapping strategy s_{rel}. In other words, the 3-tuple

[2] An element with multiple parent elements in the schema – see [21].

$\tau = < F, s_{frag}, s_{rel} >$ specifies a particular XML-to-relational mapping method. An extreme case is when S is considered as a single fragment which results in a single relation, usually with many null values. Other extreme occurs when each element in S is considered as a single fragment resulting in a huge number of relations and thus numerous join operations.

Note that fixed generic methods can be described using the 3-tuple τ as well. Each of the techniques views an XML document as general directed tree with several types of nodes. And the view can be considered as a special kind of XML schema.

In user-driven strategies the three characteristics are influenced by user-defined *annotations* which specify how a particular user wants to store selected fragments $F' = \{f'_1, f'_2, ..., f'_m\}$. The user usually provides S with *annotating attributes* from the predefined set of attribute names $\Sigma_{A'}$, which represent various fixed mapping strategies, resulting in an *annotated schema* S'. Obviously, *annotated fragments* in F' do not have to correspond to schema fragments in F. For instance, a user may specify that whole S should be stored using a 4NF decomposition and thus $F' = \{S'\}$ is a singleton, whereas corresponding F can contain several schema fragments depending on the complexity of S. Also note that while F should cover whole S, F' usually does not.

A classical user-driven strategy consists of the following steps:

1. S is provided with annotations from $\Sigma_{A'}$ resulting in S' and F'.
2. Annotated fragments from F' are decomposed into corresponding schema fragments $F_1 = \{f_1, f_2, ..., f_k\}$.
3. Not annotated fragments of S' are decomposed into schema fragments $F_2 = \{f_{k+1}, f_{k+2}, ..., f_n\}$ using a default (predefined or user-defined) fixed strategy s_{def}.
4. $F = F_1 \cup F_2$ is mapped to R using s_{frag} and s_{rel}.

Using this notation our proposed enhancing simply adds the following steps between the step 1 and 2:

a. For $\forall\ f' \in F'$ we identify a set $F'_{f'}$ of all fragments similar to f' occurring in $S' \backslash \{f'\}$.

b. For $\forall\ f' \in F'$ all fragments in $F'_{f'}$ are annotated with annotating attributes of f' and $F' = F' \cup F'_{f'}$.

c. $S' \backslash F'$ is annotated using an adaptive strategy which adds new annotated fragments to F'.

The mapping process is schematically depicted in Figure 1 where the given schema S with two annotated fragments f and g is mapped to a database schema R. If the proposed enhancing (i.e. steps 1.a – 1.c) is included, the system gradually identifies and adds new annotated fragments f_1, f_2, g_1, g_2, and g_3 which are mapped using user-required mapping strategies. If the enhancing is not included (i.e. in case of a classical user-driven strategy), only fragments f and g are annotated using user-required strategies and the rest of the schema using s_{def}.

4.1 Searching for Similar Fragments

As we have mentioned, there are numerous approaches to measuring similarity of XML data. Nevertheless, most of them cannot be directly used for our case since our demanded key characteristics differ. In particular, we search for similarity within the scope of a single schema, the similarity measure should not depend on similarity of element / attribute names but primarily on complexity of content models, and the similarity measure cannot obviously depend on the context of fragments.

Considering the problem in more depth, several fundamental questions arise:

1. How are the annotated fragments defined?
2. What types of annotations, i.e. fixed mapping strategies, are supported?
3. What measure is used for measuring similarity of two schema fragments?
4. Can we optimize the exhaustive search strategy?

The answers for the questions mutually influence each other and specify the algorithm. Furthermore, the definition of annotated fragments together with the question of their mutual intersection are closely related to supported mapping strategies.

4.1.1 Annotated Fragments

First of all, for easier processing we define a graph representation of an XML schema S, no matter if annotated or not. For easier explanation we assume that the given XML schema S is expressed in DTD[3] [9], nevertheless the algorithm can be applied to schemes expressed using, e.g., XML Schema [23] [7] language as well.

Definition 1 *A* schema graph *of an XML schema S is a directed, labelled graph $G_S = (V, E, \Sigma_E, \Sigma_A, lab)$, where*

- *V is a finite set of nodes,*
- *$E \subseteq V \times V$ is a set of edges,*
- *Σ_E is a set of element names in S,*
- *Σ_A is a set of attribute names in S, and*
- *$lab : V \rightarrow \Sigma_E \cup \Sigma_A \cup \{$"|", "*", "+", "?", ","$\} \cup \{pcdata\}$ is a surjective function which assigns a label to $\forall v \in V$.*

Definition 2 *A* fragment *of a schema S is each subgraph of S consisting of an element e, all its descendants, and corresponding edges.*

Φ is a set of all fragments of S.

Next, we assume that each annotated fragment $f' \in F'$ is uniquely determined by the element e which was annotated using an annotating attribute $a \in \Sigma_{A'}$.

Definition 3 *An* annotated element *e' of schema S' is an element provided with an annotated attribute from $\Sigma_{A'}$.*

[3] We omit supplemental constructs such as entities, CDATA sections, comments, etc.

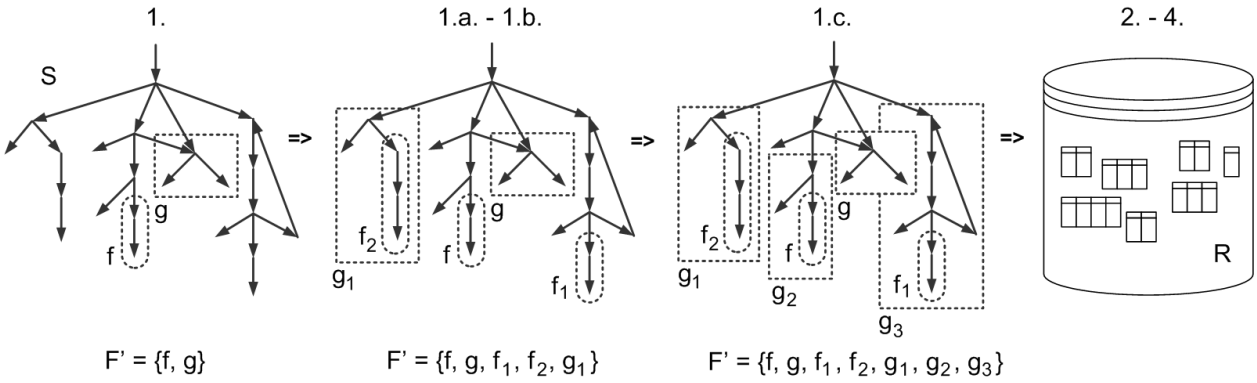

Figure 1: Schema of the mapping process

Definition 4 *An* annotated fragment f' *of schema* S' *is a fragment of* S' *rooted at an annotated element* e' *excluding all annotating attributes from* $\Sigma_{A'}$.

As we want to support shared elements and recursion, since both the constructs are widely used in real XML data [17], we must naturally allow the annotated fragments to intersect almost arbitrarily. To simplify the situation, we define an *expanded schema graph*, which exploits the idea that both the constructs purely indicate repeated occurrence of a particular pattern.

Definition 5 *An* expanded schema graph G_S^{ex} *is a result of the following transformations of schema graph* G_S*:*

1. *Each shared element is duplicated for each sharer using a* deep copy operation*, i.e. including all its descendants and corresponding edges.*
2. *Each recursive element is duplicated for each repeated occurrence using a* shallow copy operation*, i.e. only the element node itself is duplicated.*

An illustrative example of a schema graph G_S and its expanded schema graph G_S^{ex} is depicted in Figure 2. A shared element is highlighted using a dotted rectangle, a recursive element is highlighted using a dotted circle.

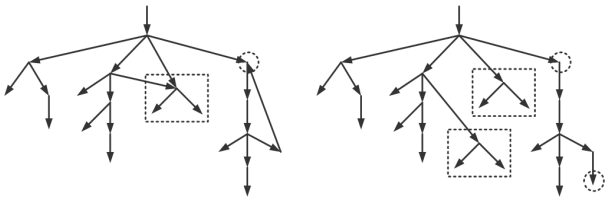

Figure 2: A schema graph G_S and an expanded schema graph G_S^{ex}

As it is obvious, in case of shared elements the expansion is lossless operation. It simply omits the key advantage of shared elements which allows reusing of previously defined schema fragments. The situation is more complicated in case of recursive elements which need to be treated in a special way henceforth. For this purpose we exploit results of statistical analysis of real-world recursive elements [17], particularly the fact that the most common type of recursion is linear[4] and that the number of repetitions of a recursive element is small, on average less than 5. The two findings enable to treat recursive elements not as elements with theoretically infinite depth, but in a much simpler way. We discuss the details later in the text.

In the following text we assume that a schema graph of an XML schema is always an expanded schema graph, if not explicitly stated alternatively.

[4]Consisting of a single recursive element that does not branch out.

4.1.2 Types of Annotations

From Definitions 4 and 5 we can easily prove the following two statements:

Lemma 1 *Each expanded schema graph* G_S^{ex} *is a tree.*

Lemma 2 *Two annotated fragments* f'_x *and* f'_y *of an expanded schema graph* $G_{S'}^{ex}$ *can intersect only if* $f'_x \subseteq f'_y$ *or* $f'_x \subseteq f'_y$.

Furthermore, we can observe that the *common schema fragment*, i.e. the intersection, contains all descendants of a particular element.

We distinguish three types of the annotation intersection depending on the way the corresponding mapping strategies influence each other.

Definition 6 *Intersecting annotations are* redundant *if the corresponding mapping strategies are applied on the common schema fragment separately.*

Definition 7 *Intersecting annotations are* overriding *if only one of the corresponding mapping strategies is applied on the common schema fragment.*

Definition 8 *Intersecting annotations are* influencing *if the corresponding mapping strategies are combined resulting in one composite storage strategy applied on the common schema fragment.*

Redundant annotations can be exploited, e.g., when a user wants to store XHTML fragments both in a single CLOB column (for fast retrieval of the whole fragment) and, at the same time, into a set of tables (to enable querying particular items). An example of overriding annotations can occur when a user specifies a general mapping strategy for the whole schema S and then annotates fragments which should be stored differently. Naturally, in this case the strategy which is applied on the common schema fragment is always the one specified for its root

element. The last mentioned type of annotations can be used in a situation when a user specifies, e.g., the 4NF decomposition for a particular schema fragment and at the same time an additional numbering schema which speeds up processing of particular types of queries. In this case the numbering schema is regarded as a supplemental index over the data stored in relations of 4NF decomposition, i.e. the data are not stored redundantly as in the first case.

We assume that each pair of annotations implemented in a corresponding system is assigned its intersection type or, if necessary, a particular combination of annotations can be denoted as *forbidden*. The existing systems [3] [5] mostly support overriding annotations, the XCacheDB system [5], in addition, supports a type of redundant intersection similar to the above described example. Furthermore, we assume that the implemented annotations are assigned priorities which specify the order in which they are composed when applied on common schema fragment.

4.1.3 Search Algorithm

The similarity measure, the search algorithm, and its possible optimization are closely related. However, the main idea of the enhancing of user-driven techniques remains the same regardless the chosen measure and algorithm. The choice of the measure influences the precision and scalability of the system, whereas the algorithm influences the efficiency of finding the required fragments. In case of "classical" adaptive methods this can be a marginal aim, since the schema adaptation is performed only once, before the schema is created. But its importance significantly rises when the system needs to be dynamic and adapt continuously.

Let us suppose that we have a similarity measure $sim(f_x, f_y) \in [0, 1]$ expressing similarity of two fragments f_x and f_y of an expanded graph, where 1 represents strong similarity and 0 strong dissimilarity, and a similarity threshold $T_{sim} \in [0, 1]$. A naive strategy would exploit an exhaustive search as depicted by Algorithm 1.

The algorithm evaluates similarity of each annotated fragment $f' \in F'$ and each fragment f_e rooted at an element $e \in S' \backslash \{f'\}$. We can assume that if the storage strategy for any f_e should not change, the user would mark it as *final*. For such fragment either the default strategy s_{def} or the strategy specified by corresponding user-defined annotation will be used, regardless the results of the search or adaptive algorithm. As such this situation is rather the problem of implementation than a theoretical one and thus we further assume that there are no such fragments in S'. On the other hand, we naturally regard fragments annotated by a user to be final by default.

The resulting similarity values are stored into so-called *similarity matrix* $\{sim[f', f_e]\}_{f' \in F',\ e \in S'}$. An element e is annotated if there exists a fragment $f_{max} \in F'$ with the highest similarity value $sim(f_{max}, f_e) > T_{sim}$. In Algorithm 1 we assume that there is always one such candidate at most. Otherwise, the system can ask for user intervention when necessary. We call this approach a *single annotation strategy (SAS)*.

Algorithm 1 Single Annotation Strategy (SAS)

Input: S', F', $sim(f_x, f_y)$, T_{sim}
Output: F'', i.e. $F' \cup$ newly annotated fragments

```
    {construction of the similarity matrix}
 1: F'' ← F'
 2: for all f' ∈ F' do
 3:    for all element e ∈ S' do
 4:       f_e ← subgraph rooted at e
 5:       if e ∈ S'\{f'} then
 6:          sim[f', f_e] ← sim(f', f_e)
 7:       else
 8:          sim[f', f_e] ← 0
 9:       end if
10:    end for
11: end for
    {annotation strategy}
12: for all element e ∈ S' do
13:    max_e ← max_{f'∈F'}{sim[f', f_e]}
14:    f_max ← f' ∈ F' s.t. sim[f', f_e] = max_e
15:    if max_e > T_sim then
16:       f_e.annotation ← f_max.annotation
17:       F'' ← F'' ∪ {f_e}
18:    end if
19: end for
20: return F''
```

The question is whether this approach is correct actually. From another point of view it could be more reasonable to annotate an element e using annotations of all fragments $f' \in F'$ s.t. $sim(f', f_e) > T_{sim}$. Together with the assumption that for each pair of annotations the result of their composition is predefined and that the annotations have priorities according to which they are composed, this approach seems to be a better choice since it does not omit important information. But, on the other hand, let us consider the situation depicted in Figure 3, where for $i \in \{1, 2, 3\}$ $sim(f', f_i) > T_{sim}$ and f' is the annotated fragment.

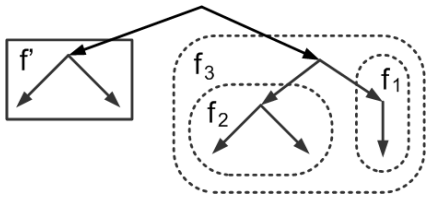

Figure 3: Similar fragments on the same root path

The problem is whether we can annotate all the three fragments f_1, f_2, f_3 using the annotation of f', especially what will be the result of intersection in case of f_1 and f_3 or f_2 and f_3, i.e. fragments occurring on the same *root path*[5]. We can naturally assume that intersection of two identical annotations is overriding and as such has no effect. Thus we could annotate only the topmost fragment on each root path. In case of example in Figure 3 this rule would be applied twice, resulting in a single annotation of fragment f_3. But what if we knew, in addition, that $sim(f', f_1) > sim(f', f_3)$ and $sim(f', f_2) > sim(f', f_3)$? As it is obvious, in such case it is seems to be more reasonable and natural to annotate fragments f_1 and f_2 rather than whole f_3. Or this

[5] A path from the root node to a leaf node.

situation can be again a case for user intervention, depending on the point of view of it. We will further consider the former one.

If we generalize the idea, the algorithm annotates an element e using annotations of all fragments $f' \in F'$ s.t. $sim(f', f_e) > T_{sim}$ and $\nexists$ element e' on any root path traversing e s.t. $sim(f', f_{e'}) > sim(f', f_e)$. The resulting algorithm, so-called *multiple annotation strategy (MAS)*, is depicted by Algorithm 2, where $e.ancs$ denotes a set of (direct or undirect) ancestors of element e and $e.descs$ denotes a set of (direct or undirect) descendants of e. The process of construction of the similarity matrix remains the same as in case of Algorithm 1.

Algorithm 2 Multiple Annotation Strategy (MAS)

Input: S', F', $sim(f_x, f_y)$, T_{sim}
Output: F'', i.e. $F' \cup$ newly annotated fragments

```
1:  F'' ← F'
    {construction of the similarity matrix}
2:  -//-
    {annotation strategy}
3:  for all f' ∈ F' do
4:    for all element e ∈ S' do
5:      if (sim[f', f_e] > T_sim) ∧
        (∄e_a ∈ e.ancs : sim[f', f_ea] > sim[f', f_e]) ∧
        (∄e_d ∈ e.descs : sim[f', f_ed] > sim[f', f_e])
        then
6:        f_e.annotation ← f'.annotation
7:        F'' ← F'' ∪ {f_e}
8:      end if
9:    end for
10: end for
11: return F''
```

Using this approach we should consider what will happen in case a user annotates two structurally identical (or too similar) fragments using different annotations. We cannot simply rely on predefined type of their intersection and corresponding priorities, because the situation is a slightly different one. In this case the system should rather ask for user intervention whenever it is not able to decide. And this is again a problem of the particular implementation.

4.1.4 Similarity Measure and Optimization of the Search Algorithm

Now let us consider the search strategy from the point of view of complexity of the algorithm. Figure 4 depicts an example of processing a single annotated fragment, in particular the amount of similarity comparisons. Annotated fragments f and g are highlighted using rectangles, all schema fragments, which are compared with f are highlighted using dotted ovals.

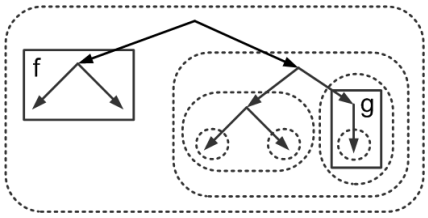

Figure 4: Exhaustive search strategy

If we do not know any features of the measure, there are not many ways how to avoid the exhaustive search. Also the order in which fragments in F' are processed is then unimportant. But although we can assume that $card(F') = m$ is small, i.e. that a user annotates several fragments but the number is not large, the exhaustive search can be expensive due to the size of G_S^{ex}. And even from the simple example in Figure 4 it is obvious that there are pairs of schema fragments which do not have to be compared at all. Another problem is the complexity of the if condition of Algorithm 2 (line 5) which can in the worst case lead to multiple searching through the whole G_S^{ex}. So in both the cases we need to avoid the unnecessary similarity evaluations.

It seems promising to borrow the idea of clustering, similarly to paper [22], where the distance between schema fragments is determined by their mutual similarity, e.g. $dist(f_x, f_y) = 1 - sim(f_x, f_y)$. An example is depicted in Figure 5 for the sample schema in Figure 4.

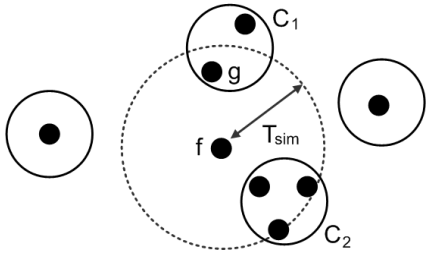

Figure 5: Exploitation of clustering

All schema fragments (depicted using black-filled circles) are divided into clusters C_1, C_2,..., C_k (depicted using black circular lines) having their centroids c_1, c_2,..., c_k and radii r_1, r_2,..., r_k (or one common radius r, depending on the implementation). Having this information, only those schema fragments have to be compared with fragment f, whose clusters intersect the cluster with centroid f and radius T_{sim}. In case of Figure 5 these are clusters C_1 and C_2. Obviously, if the clusters were selected appropriately, the amount of comparisons would decrease rapidly. Hence the key concern of all clustering algorithms is mainly the construction of the clusters.

The construction is usually performed using a *k-means algorithm* or its variations (e.g. [22]), where the initial clusters are selected randomly and then iteratively improved. In the i-th iteration each fragment is compared with centroids of all clusters and assigned to the closest one. The algorithm terminates if none of the clusters changes, otherwise new centroids are computed and $(i+1)$-th iteration follows. The complexity of the construction is $O(I \cdot |\Phi| \cdot k)$, where I is the number of iterations. In case of complexity of similarity evaluation the worst case is when either $k = 1$ or all k clusters mutually intersect, i.e. when we cannot avoid any of the similarity comparisons. Hence in the worst case the number of comparisons is the same as in the exhaustive search strategy and the complexity can worsen only the pre-processing, i.e. the construction of clusters. And this is the step we want to remove too.

For further optimization we can exploit characteristics of the chosen similarity measure. The existing algorithms for measuring similarity on schema level usually

exploit various supplemental *matchers* [20], i.e. functions which evaluate similarity of a particular feature of the given schema fragments, such as, e.g., similarity of number and types of leaf nodes, similarity of root element names, similarity of context, etc.

Definition 9 *A* matcher *is a function* $m : \Phi^2 \rightarrow [0,1]$ *which evaluates similarity of a particular feature of two schema fragments* $f_x, f_y \in \Phi$.

Definition 10 *A* partial similarity measure *is a function* $m_{part} : \Phi^2 \rightarrow [0,1]^p$ *which evaluates similarity of the given schema fragments* $f_x, f_y \in \Phi$ *using matchers* $m_1, m_2, ..., m_p : \Phi^2 \rightarrow [0,1]$ *and returns a p-tuple of their results.*

Then the partial results are combined using an appropriate approach, usually a kind of a weighted sum, into the resulting composite similarity value.

Definition 11 *A* composite similarity measure *is a function* $m_{comp} : [0,1]^p \rightarrow [0,1]$ *which combines the results of particular matchers and returns the total similarity value.*

Due to the features of selected partial matchers the existing techniques usually exploit a bottom-up strategy, i.e. starting from leaf nodes towards the root node. Together with the previously mentioned problem of similar intersecting fragments this is why we need to know the behavior of the similarity measure on particular root paths.

For instance, if we knew that the similarity measure is concave, i.e. that it has only one global maximum, we could skip processing of all the ancestors on the current root path whenever we reach the fragment with the extreme value. A sample situation can be seen in Figure 6 which depicts an example of a graph of similarity function for an annotated fragment f' and fragments $f_1, f_2, ..., f_r$ on a single root path. From the graph we can see, that only fragments f_1, f_2, f_3, f_4 need to be processed (f_4 for testing the extremity), then the similarity evaluation can terminate, skipping fragments $f_5, f_6, ..., f_r$.

As it is obvious, this way we can decrease the number of unnecessary similarity evaluations as well as avoid pre-processing of the schema and expensive checking of the if condition of Algorithm 2. Naturally, the efficiency of such approach depends strongly on the position of the extreme on the root path. The key problem is how to define such similarity measure. For our purpose we need a measure which focuses especially on the structure of the compared fragments, known equivalences or relations between regular expressions, differences between simple and complex types, etc., i.e. on features that influence the efficiency of database processing the most. But it is hard, if not impossible, to propose a measure with concave behavior which is at the same time enough precise in relation to these requests. Nevertheless, we can exploit a relaxed version of this idea as a kind of heuristic of the bottom-up strategy.

Although we can hardly ensure that m_{comp} is concave, we can assume that at least q of the matchers, where $1 \leq q \leq p$, have this property. Without loss of generality we suppose that these are $m_1, m_2, ..., m_q$. For instance a trivial matcher with such behavior can compare the number of distinct element or attribute names, the amount of similar operators, the depth of the corresponding regular expression, etc. The heuristic is then based on the idea that if at least "sufficient amount" of the q matchers exceed their extreme value, we can terminate processing of the current root path too. There are just two differences from the previously mentioned idea. Firstly, the exceeding of the particular extreme is expressed using a threshold $T_{ex} \in [0,1]$ which guarantees that processing of the current root path does not terminate neither too soon (to reach the optimum of the composite similarity measure) nor too late (to avoid unnecessary similarity evaluations). Secondly, as it is obvious, the matchers themselves are not precise in terms of similarity evaluation. Hence each of them is assigned a user-specified *reliability* $r_1, r_2, ..., r_q \in [0,1]$, where 0 expresses strong unreliability and 1 strong reliability of the matcher. The reliabilities then influence the real value of threshold T_{ex} for particular matchers multiplying its value.

The whole optimization of the approach, so-called *basic annotation strategy (BAS)*, is depicted by Algorithm 3, where function *terminate* returns *true* if the search algorithm should terminate in the given node, otherwise it returns *false*. Furthermore, we assume that each element of the graph is assigned an auxiliary list of *candidates* consisting of pairs $< fragment, similarity >$, i.e. references to fragments (and corresponding similarity values) within its subtree that are candidates for annotation.

The algorithm processes schema graph starting from leaf nodes. For each root path the optimal similarity value and the reference to corresponding fragment are propagated until a better candidate is found or the condition of the heuristic is fulfilled. Then the processing of the current root path is terminated and current candidates are annotated. The complexity of the algorithm depends on the heuristics. In the worst case it does not enable to skip processing of any node that results in the exhaustive search. But in contrast to the clustering approach we have avoided the expensive preprocessing of the algorithm.

In general we could use an arbitrary similarity measure, not exactly the above defined composite one. It is also possible to use disjoint sets of matchers for the heuristic and for the composite similarity measure. Nevertheless, we will deal with the above described ones, since it is the typical and verified way for evaluating similarity among XML schemes.

4.1.5 Recursive Elements

Last but not least, we have to solve the open problem of expanded recursive elements, since the expansion is not a lossless operation as in case of shared elements. As we have already indicated, we will exploit the results of analysis of real-world XML data which shows two important aspects [17]:

1. Despite it is generally believed that recursive elements are of marginal importance, they are used in a significant portion of real XML data.

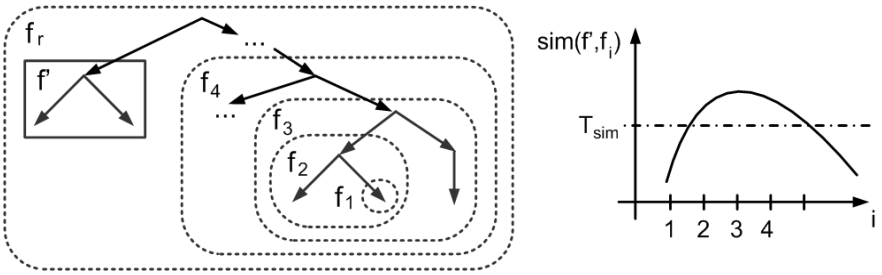

Figure 6: Exploitation of behavior of similarity function

2. Although the recursive elements can have arbitrarily complex structure, the most common type of recursion is linear and the average depth of recursion is low.

If we realize that we need the "lost" information about recursion only at one stage of the algorithm, the solution is quite obvious. We analyze the structure of schema fragments when evaluating matchers $m_1, m_2, ..., m_p$, whereas each of the matchers describes similarity of a particular feature of the given fragments. In case the fragments contain recursive elements we will not use the exact measure, but its approximation with regard to the real complexity of recursive elements. For instance if the matcher analyzes the maximum depth of fragment containing a recursive element, the resulting depth is not infinite, but considers the average depth of real-world recursive elements.

The question is whether it is necessary to involve a matcher which analyzes the amount of recursive elements in schema fragments. On one hand it can increase the precision of the composite measure. But from another point of view the approximation transforms the recursive element to a "classical" element and hence such matcher can be misleading.

4.2 Adaptive Mapping Strategy

At this stage of the algorithm we have a schema S' and a set of annotated fragments F' which involve the user-defined fragments and fragments identified by Algorithm 3. As the second enhancing we want to apply an adaptive mapping strategy to the remaining parts of the schema.

As mentioned previously, the key idea of adaptive strategies is to adapt the target relational schema R to the expected future application, which is specified by a sample set of XML data and/or XML queries. The techniques define a set of XML-to-XML transformations which produce a space of equivalent or more general XML schemes. A search algorithm is used to find a schema, where the evaluation of the given queries is most efficient. Thus at first glance the user-driven techniques have nothing in common with the adaptive ones. Or, we could expect that the user provides not only a set of annotations, but also sample XML documents and XML queries. Then we could use a classical adaptive strategy for not annotated parts of schema S'. Such combination should not be difficult considering the fact that user-driven techniques enable to store various schema fragments in various ways. Nevertheless, the key shortcoming of such approach is that the user is expected to provide too many information.

Under a closer investigation we can see that the user-given annotations provide a similar information – they say how particular schema fragments should be stored to enable efficient data querying and processing. Thus we can simply reuse the user-given information. For this purpose we define an operation *contraction* which enables to omit those schema fragments, where we already know the storage strategy, and focus on the remaining ones.

Definition 12 *A* contraction *of an expanded schema graph $G^{ex}_{S'}$ with annotated fragment set F' is an operation which replaces each fragment $f' \in F'$ with a single auxiliary node called a* contracted node.

The resulting graph is called a contracted graph $G^{con}_{S'}$.

From Definitions 4 and 12 we can easily prove the following simple statement which enables to reuse the search algorithm on the contracted graph.

Lemma 3 *Contracted graph $G^{con}_{S'}$ of a connected expanded schema graph $G^{ex}_{S'}$ is connected.*

The basic idea of the adaptive strategy is as follows: Having a contracted graph $G^{con}_{S'}$ we repeat the search algorithm (in particular its slight modification) and operation contraction until there can be found any fragment to annotate. Contrary to the previous situation the search algorithm has the following differences:

- It searches for schema fragments which are not involved in the schema, i.e. it searches among all nodes of the given (contracted) graph and returns the (eventually empty) set of found fragments.
- For similarity evaluation we do not take into account contracted nodes, i.e. neither their current, nor their previous structure or any other features. These were already analyzed and processed in previous steps of the algorithm.
- The annotations of contracted nodes are always overriding in relation to the newly defined ones.
- The required threshold is more precise, i.e. we use a threshold $T_{con} < T_{sim}$.

We denote this modification of BAS as a *contraction-aware annotation strategy (CAS)*. (We omit its formal description for obviousness and the paper length.) The resulting annotating strategy, so called *global annotation strategy (GAS)*, is depicted by Algorithm 4, where function *contract* applies operation contraction on expanded graph of the given schema and corresponding fragments,

Algorithm 3 Basic Annotation Strategy (BAS)

Input: S', F', m_1, m_2, ..., m_q, m_{q+1}, ..., m_p, r_1, r_2, ..., r_q, T_{ex}, m_{comp}, T_{sim}
Output: F'', i.e. $F' \cup$ newly annotated fragments

1: $F'' \leftarrow F'$
2: **for all** $f' \in F'$ **do**
3: listToProcess $\leftarrow$ leaf elements of $G^{ex}_{S'} \backslash \{f'\}$
4: listOfProcessed $\leftarrow \emptyset$
5: **while** listToProcess $\neq \emptyset$ **do**
6: **for all** $e \in$ listToProcess **do**
7: e.candidates $\leftarrow \emptyset$
8: $f_e \leftarrow$ subgraph rooted at e
9: $sim_e \leftarrow m_{comp}(f', f_e)$
10: **for all** $c \in e$.subelems **do**
11: **for all** $< f, sim > \in c$.candidates **do**
12: **if** $sim > sim_e$ **then**
13: e.candidates $\leftarrow e$.candidates $\cup \{< f, sim >\}$
14: **end if**
15: **end for**
16: **end for**
17: **if** e.candidates $= \emptyset \wedge sim_e > T_{sim}$ **then**
18: e.candidates $\leftarrow e$.candidates $\cup \{< f_e, sim_e >\}$
19: **end if**
20: **if** terminate(f', e, m_1, m_2, ..., m_q, r_1, r_2, ..., r_q, T_{ex}) **then**
21: **for all** $< f, sim > \in e$.candidates **do**
22: f.annotation $\leftarrow f'$.annotation
23: $F'' \leftarrow F'' \cup \{f\}$
24: **end for**
25: **else**
26: **if** $\forall\ s \in e$.siblings : $s \in$ listOfProcessed **then**
27: listToProcess $\leftarrow$ listToProcess $\cup \{e$.parent$\}$
28: **end if**
29: **end if**
30: listToProcess $\leftarrow$ listToProcess $\setminus \{e\}$
31: listOfProcessed $\leftarrow$ listOfProcessed $\cup \{e\}$
32: **end for**
33: **end while**
34: **end for**
35: **return** F''

Algorithm 4 Global Annotation Strategy (GAS)

Input: S', F', m_1, m_2, ..., m_q, m_{q+1}, ..., m_p, r_1, r_2, ..., r_q, T_{ex}, m_{comp}, T_{sim}, T_{con}
Output: F'', i.e. $F' \cup$ newly annotated fragments

1: $F'' \leftarrow$ BAS(S', F', m_1, m_2, ..., m_p, r_1, r_2, ..., r_q, T_{ex}, m_{comp}, T_{sim})
2: $F^{tmp} \leftarrow F''$
3: **while** $F^{tmp} \neq \emptyset$ **do**
4: contract(S', F^{tmp})
5: $F^{tmp} \leftarrow$ CAS(S', F', m_1, m_2, ..., m_p, r_1, r_2, ..., r_q, T_{ex}, m_{comp}, T_{con})
6: $F'' \leftarrow F'' \cup F^{tmp}$
7: **end while**
8: expand_contractions(S', F'')
9: **return** F''

and function *expand_contractions* expands all the contracted nodes of the given schema to the original ones.

The resulting complexity of the algorithm depends on the number of iterations of the cycle (lines 3 – 7). In the worst case each iteration results in annotating of a single element, i.e. the search algorithm repeats $(|\Phi| - |F'| + 1)$ times.

Considering the whole approach, we are especially interested in the efficiency of the resulting XML-to-relational storage strategy which can be verified only through appropriate tests on real XML data. We discuss it in the following section.

5 Open Issues

In the previous section we have described and discussed the proposed algorithm on theoretical level. We have mentioned several possible solutions and discussed their consequences and disadvantages as well as reasons for the choices we have made. But despite the detailed description there are still several open issues. We distinguish two main categories:

1. Features of the particular implementation and
2. Behavior of the algorithm on real XML data.

As for the former case the key implementation decisions are especially:

- the set of supported schema annotations, types of their mutual intersection (or forbiddance), and their priorities,

- the matchers m_1, m_2, ..., m_q, m_{q+1}, ..., m_p and corresponding reliabilities r_1, r_2, ..., r_q,
- the composite similarity measure m_{comp}, and
- the thresholds T_{sim}, T_{ex}, and T_{con}.

The key problem lies especially in tuning of reliabilities and thresholds, since both influence the precision and efficiency of the system strongly. The remaining characteristics are related rather to its usefulness and versatility. There are also marginal questions such as, e.g., whether the system will support final elements or user intervention if there are more candidates for a particular situation. But these features do not have the key influence on the proposed approach itself.

The latter category of open issues is quite unpredictable, despite the existing statistics of real XML data [17]. It is caused mainly by two facts. Firstly, although we know the usual characteristics of the real data, we cannot predict especially the behavior of more complex similarity measures due to the above mentioned tuning of the characteristics. And secondly, we cannot predict the behavior of the proposed adaptive strategy, since we have no information about the structure of contracted graphs of real data. Furthermore, the choice of particular schema fragments will be strongly related to the type of the tested data and thus the efficiency of the resulting storage strategy can vary remarkably.

As it is obvious, both the categories are also related significantly. And though some particular features can be estimated or preset according to the existing user-driven systems and statistical analysis of data, most of them will still require a series of experimental tests.

6 Conclusion

The main aim of this paper was to illustrate that since the idea of database-based XML processing methods is still up-to-date, the techniques should and can be further enhanced. On this account we have proposed a user-driven mapping algorithm which is able to exploit the user given information, i.e. schema annotations, more deeply and, at the same time, to find the mapping strategy for the not annotated parts more efficiently – using an adaptive approach. We have described and discussed the algorithm on theoretical level as well as summed up the corresponding open issues related to particular implementation decisions.

A natural next step of our work is the implementation of the proposed algorithm and especially experimental tests of its behavior on real data. For this purpose we are enhancing our previous implementation of a fixed XML-to-relational mapping strategy [14], which exploits object-oriented features of XML Schema language together with features of object-relational database systems. For the testing we will exploit information about categories of real XML data and their typical features [17] as well as existing related works which could help with tuning the system.

A possible further enhancing of our approach can be found in focussing on statistically frequent XML schema patterns. They can be used in several ways – e.g. as a reasonable default setting or as XML patterns with a higher priority.

Another improvement can be an exploitation of the semantic of element and/or attribute names. The similarity can be searched not only on structural level, but using a kind of thesaurus or appropriate user-given information. Although our proposal focuses mainly on structural similarities related to efficiency of database processing, it is at least worth testing whether the semantic of the names carries additional important information useful for this purpose too.

Next interesting task could be also a combination of our approach with a real cost-driven one, i.e. an exploitation of both user-given annotations and a sample set of XML data and XML queries together. As we have already mentioned, the key disadvantage is in the amount of required input data. But, on the other hand, the combination of the two approaches could bring interesting results or the efficiency of the two approaches can be at least compared.

And last but not least, the key enhancing lies in dynamic adaptability of the system [15]. This challenging but non-trivial task would solve the remaining disadvantage of the adaptive methods – the fact that the schema is adapted only once, at the beginning but not in case the application changes.

References

[1] *XHTML 1.0 The Extensible HyperText Markup Language (Second Edition)*. W3C Recommendation, August 2002. `http://www.w3.org/TR/xhtml1/`.

[2] S. Amer-Yahia. *Storage Techniques and Mapping Schemas for XML*. Technical Report TD-5P4L7B, AT&T Labs-Research, 2003.

[3] S. Amer-Yahia, F. Du, and J. Freire. A Comprehensive Solution to the XML-to-Relational Mapping Problem. In *WIDM'04: Proceedings of the 6th Annual ACM International Workshop on Web Information and Data Management*, pages 31–38, New York, NY, USA, 2004. ACM Press.

[4] S. Amer-Yahia and M. Fernandez. *Overview of Existing XML Storage Techniques*. AT&T Labs-Research, 2001.

[5] A. Balmin and Y. Papakonstantinou. Storing and Querying XML Data Using Denormalized Relational Databases. *The VLDB Journal*, 14(1):30–49, 2005.

[6] E. Bertino, G. Guerrini, and M. Mesiti. A Matching Algorithm for Measuring the Structural Similarity between an XML Document and a DTD and its Applications. *Inf. Syst.*, 29(1):23–46, 2004.

[7] P. V. Biron and A. Malhotra. *XML Schema Part 2: Datatypes (Second Edition)*. W3C Recommendation, October 2004. `www.w3.org/TR/xmlschema-2/`.

[8] P. Bohannon, J. Freire, P. Roy, and J. Simeon. From XML Schema to Relations: A Cost-based

Approach to XML Storage. In *ICDE '02: Proceedings of the 18th International Conference on Data Engineering*, pages 64–75, Washington, DC, USA, 2002. IEEE Computer Society.

[9] T. Bray, J. Paoli, C. M. Sperberg-McQueen, E. Maler, and F. Yergeau. *Extensible Markup Language (XML) 1.0 (Fourth Edition)*. W3C Recommendation, September 2006. `http://www.w3.org/TR/REC-xml/`.

[10] H. H. Do and E. Rahm. COMA – A System for Flexible Combination of Schema Matching Approaches. In *VLDB'02: Proceedings of the 28th International Conference on Very Large Data Bases*, pages 610–621, Hong Kong, China, 2002. Morgan Kaufmann Publishers Inc.

[11] D. Florescu and D. Kossmann. Storing and Querying XML Data using an RDMBS. *IEEE Data Eng. Bull.*, 22(3):27–34, 1999.

[12] M. Klettke and H. Meyer. XML and Object-Relational Database Systems – Enhancing Structural Mappings Based on Statistics. In *Selected papers from the Third International Workshop WebDB 2000 on The World Wide Web and Databases*, pages 151–170, London, UK, 2001. Springer-Verlag.

[13] J. Madhavan, P. A. Bernstein, and E. Rahm. Generic Schema Matching with Cupid. In *VLDB '01: Proceedings of the 27th International Conference on Very Large Data Bases*, pages 49–58, San Francisco, CA, USA, 2001. Morgan Kaufmann Publishers Inc.

[14] I. Mlynkova and J. Pokorny. From XML Schema to Object-Relational Database – an XML Schema-Driven Mapping Algorithm. In *ICWI'04: Proceedings of IADIS International Conference WWW/Internet*, pages 115–122, Madrid, Spain, 2004. IADIS.

[15] I. Mlynkova and J. Pokorny. Adaptability of Methods for Processing XML Data using Relational Databases – the State of the Art and Open Problems. In *RCIS'07: Proceedings of the 1st International Conference on Research Challenges in Information Science (to appear)*, Ouarzazate, Morocco, April 2007.

[16] I. Mlynkova and J. Pokorny. *Exploitation of Similarity and Pattern Matching in XML Technologies*. Technical report 2006/13. Charles University, Prague, Czech Republic, November 2006. `http://kocour.ms.mff.cuni.cz/~mlynkova/doc/tr2006-13.pdf`.

[17] I. Mlynkova, K. Toman, and J. Pokorny. Statistical Analysis of Real XML Data Collections. In *COMAD'06: Proceedings of the 13th International Conference on Management of Data*, pages 20–31, New Delhi, India, 2006. Tata McGraw-Hill Publishing Company Limited.

[18] P. K.L. Ng and V. T.Y. Ng. Structural Similarity between XML Documents and DTDs. In *ICCS'03: Proceedings of the International Conference on Computational Science*, pages 412–421. Springer Berlin / Heidelberg, 2003.

[19] A. Nierman and H. V. Jagadish. Evaluating Structural Similarity in XML Documents. In *WebDB'02: Proceedings of the Fifth International Workshop on the Web and Databases*, pages 61–66, Madison, Wisconsin, USA, 2002.

[20] E. Rahm and P. A. Bernstein. A Survey of Approaches to Automatic Schema Matching. *The VLDB Journal*, 10(4):334–350, 2001.

[21] 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: Proceedings of 25th International Conference on Very Large Data Bases*, pages 302–314, San Francisco, CA, USA, 1999. Morgan Kaufmann Publishers Inc.

[22] M. Smiljanic, M. van Keulen, and W. Jonker. Using Element Clustering to Increase the Efficiency of XML Schema Matching. In *ICDEW'06: Proceedings of the 22nd International Conference on Data Engineering Workshops*, pages 45–54, Los Alamitos, CA, USA, 2006. IEEE Computer Society Press.

[23] H. S. Thompson, D. Beech, M. Maloney, and N. Mendelsohn. *XML Schema Part 1: Structures (Second Edition)*. W3C Recommendation, October 2004. `www.w3.org/TR/xmlschema-1/`.

[24] W. Xiao-ling, L. Jin-feng, and D. Yi-sheng. An Adaptable and Adjustable Mapping from XML Data to Tables in RDB. In *Proceedings of the VLDB'02 Workshop EEXTT and CAiSE 2002 Workshop DTWeb*, pages 117–130, London, UK, 2003. Springer-Verlag.

[25] Z. Zhang, R. Li, S. Cao, and Y. Zhu. Similarity Metric for XML Documents. In *FGWM03: Proceedings of Workshop on Knowledge and Experience Management*, Karlsruhe, Germany, 2003.

[26] S. Zheng, J. Wen, and H. Lu. Cost-Driven Storage Schema Selection for XML. In *DASFAA'03: Proceedings of the 8th International Conference on Database Systems for Advanced Applications*, pages 337–344, Kyoto, Japan, 2003. IEEE Computer Society.