Adaptability of Methods for Processing XML Data using Relational Databases – the State of the Art and Open Problems
(Technical Report)

Irena Mlynkova and Jaroslav Pokorny

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

Abstract. XML technologies have undoubtedly become a standard for data representation and manipulation and are widely used in various spheres of human activities. Thus it is inevitable to propose and implement efficient techniques for managing and processing XML data.
A natural alternative is to exploit tools and functions offered by relational database systems. Unfortunately this approach has many opposers who point out especially its inefficiency caused by structural differences between XML data and relations. On the other hand, relational databases have long theoretical and practical history and represent a mature and reliable technology, i.e. they can offer properties that no native XML database can offer yet. On this account we believe that the database-based XML processing should be further studied and enhanced.
In this paper we study techniques, which enable to improve XML processing based on relational databases, so-called *adaptive* or *flexible mapping methods*. First of all, we discuss reasons why these techniques are important and promising. Secondly, we provide an overview of existing approaches, we classify their main features, and sum up the most important findings and characteristics. Finally, we discuss possible improvements and corresponding key problems.

1 Introduction

Without any doubt the eXtensible Markup Language (XML) [11] is currently one of the most popular formats for data representation. It is well-defined, easy-to-use, and involves number of other "side" recommendations such as languages for structural specification, transformation, querying, updating, etc. The wide popularity naturally invoked an enormous endeavor to propose faster and more efficient methods and tools for managing and processing XML data. Soon it was possible to distinguish several different directions based on various storage strategies. The four most popular ones are:

– methods which store XML data in a file system,
– methods which store and process XML data using an (object-)relational database system,
– methods which exploit a pure object-oriented approach, and
– native methods that use special indices and/or data structures proposed particularly for XML data processing.

Naturally, each of these approaches has both keen supporters and opposers who emphasize its particular advantages or disadvantages. The situation is not good especially for file system-based and pure object-oriented methods. The former ones suffer from inability of querying without any additional preprocessing of the data, while the latter approach fails especially in finding a corresponding efficient and comprehensive tool. Expectably, the highest-performance techniques are the native ones since they are proposed particularly for XML processing and do not need to adapt existing structures to a new purpose. Nevertheless, the most practically used ones are undoubtedly methods which exploit features of (object-) relational databases. The reason is that such databases are still regarded as universal and powerful data processing tools and in relation to their long theoretical and practical history they 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 especially their possible improvements should be studied and further enhanced.

Under a closer investigation the database-based[1] methods can be further classified and analyzed [21]. We usually distinguish:

– *generic methods*, i.e. methods which store XML data regardless its possibly existing schema (e.g. [12]),
– *schema-driven methods*, i.e. methods based on structural information from existing schema of XML documents (e.g. [29]), and
– *user-defined methods*, i.e. methods which leave all the storage decisions in hands of future users (e.g. [1]).

Techniques of the first mentioned type usually view an XML document as a general directed tree with several types of nodes. The fact that they do not exploit XML schemes can be regarded as both advantage and disadvantage. On one hand they do not depend on its existence but, on the other hand, they cannot exploit the additional structural or type information. But together with the finding that a significant portion of real XML documents (52% [19] of randomly crawled or 7.4% [22] of semi-automatically collected[2]) have no schema at all, they seem to be the most practical choice.

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

[2] Data collected with interference of a human operator who removes damaged, artificial, too simple, and/or otherwise useless XML data.

By contrast, schema-driven methods have contradictory advantages and disadvantages. Considering the disadvantages the situation is even worse for methods which are based particularly on XML Schema [31] [8] definitions (XSDs) and focus on their special features [20]. As it is expectable, XSDs are used even less (only for 0.09% [19] of randomly crawled or 38% [22] of semi-automatically collected XML documents) and even if they are used, they often (in 85% of cases [7]) define so-called *local tree grammars* [24], i.e. languages that can be defined using DTD as well. The most exploited "non-DTD" features are usually simple types whose lack in DTD is well-known and crucial but for XML data processing have only a side optimization effect.

Another problem of purely schema-driven methods is that information XML schemes provide is not satisfactory. Analysis of both XML documents and XML schemes together [22] shows that XML schemes are too general in comparison to their instances. Excessive examples can be recursion or "*" operator which allow infinitely deep or wide XML documents. Naturally XML schemes also cannot provide any information about retrieval frequency of an element or attribute. Thus not only XML schemes but also corresponding XML documents and XML queries need to be taken into account to get overall notion of the demanded XML-processing application.

The last mentioned type of approach, i.e. the user-defined one, is a bit different. It does not involve methods for automatic database storage but rather tools for specification of the target database schema and required XML-to-relational mapping. Though it seems to be a marginal approach, it is commonly offered by almost all known (object-)relational database systems [2] as a feature that enables users to define what suits them most instead of being restricted by features and especially disadvantages of a particular technique. Nevertheless, the disadvantage is evident – it assumes that the user is skilled in both database and XML technologies. And particularly for complex applications the task to propose a good database schema is not easy.

As we can observe, advantages of all three approaches are closely related to the particular situation. Thus it seems to be advisable to propose a method which is able to exploit the current situation and information or at least to comfort to them. Naturally this idea is not a brand new one. If we analyze database-based methods more deeply, we can distinguish so-called *flexible* or *adaptive* methods (e.g. [15], [10], [32], [34], [4], [6]). They take into account a given sample set of XML data, XML queries, and/or other various user-given information which specify the future usage and adapt the resulting database schema to them. It is not surprising that such techniques have better performance results than the *fixed* ones (e.g. [12], [29], [20]), i.e. methods which use predefined set of mapping rules and heuristics regardless the intended future use. Nevertheless, the adaptive techniques have also one great disadvantage – the fact that the target database schema is adapted only once, at the beginning. Thus if the expected usage strategy changes, the efficiency of such techniques can be even worse than in case of the corresponding fixed ones. Consequently the adaptability needs to be dynamic.

The idea to adapt a technique to a sample set of data is also closely related to analyses of typical features and properties of real XML documents [22]. If we combine these two ideas, we can assume that a method which focuses especially on these typical XML features will be also more efficient than the general one. A similar observation is already widely exploited for example in techniques which represent XML documents as a set of points in multidimensional space [16] [17]. Efficiency of such techniques depends strongly on depth of XML documents or number of distinct paths – naturally both the values should be as small as possible. Fortunately XML analyses confirm that real XML documents are surprisingly shallow – the average depth does not exceed 10 levels [19] [22].

Generally speaking the presumption that an adaptive enhancing of XML-processing methods focusing on given or typical situations is undoubtedly a promising type of improvement. In this paper we study these techniques from various points of view. We provide an overview of existing approaches, we classify them and their main features, and we sum up the most important findings and characteristics. Finally, we discuss possible improvements and corresponding key problems.

The rest of the paper is structured as follows: The second section contains a brief introduction to formalism used throughout the paper. Section 3 describes and classifies the existing related works, both practical and theoretical and Section 4 sums up their main characteristics. Section 5 discusses possible ways of improvement of the recent approaches and finally, the sixth section provides conclusions.

2 Definitions and Formalism

Before we begin to describe and classify adaptive methods, we state several basic terms used in the rest of the text.

An *XML document* is usually viewed as a directed labeled tree with several types of nodes whose edges represent relationships among them. Side structures, such as entities, comments, CDATA sections, processing instructions, etc., are without loss of generality omitted.

Definition 1. An XML document *is a directed labeled tree* $T = (V, E, \Sigma_E, \Sigma_A, \Gamma, lab, r)$*, where*

- V *is a finite set of nodes,*
- $E \subseteq V \times V$ *is a set of edges,*
- Σ_E *is a finite set of element names,*
- Σ_A *is a finite set of attribute names,*
- Γ *is a finite set of text values,*
- $lab : V \to \Sigma_E \cup \Sigma_A \cup \Gamma$ *is a surjective function which assigns a label to each* $v \in V$*, whereas* v *is an element if* $lab(v) \in \Sigma_E$*, an attribute if* $lab(v) \in \Sigma_A$*, or a text value if* $lab(v) \in \Gamma$*, and*
- r *is the root node of the tree.*

A *schema* of an XML document is usually described using DTD or XML Schema language. Both the languages use a similar approach and describe the allowed structure of an element using its *content model*. An XML document is *valid* against a schema if each element matches its content model. (We will state the definitions for DTDs only. For XSDs are often used similar ones involving non-DTD structures. We omit them for the paper length.)

Definition 2. A content model α *over a set of element names* Σ'_E *is a regular expression defined as* $\alpha = \epsilon \mid pcdata \mid f \mid (\alpha_1, \alpha_2, ..., \alpha_n) \mid (\alpha_1|\alpha_2|...|\alpha_n) \mid \beta* \mid \beta+ \mid \beta?$, *where* ϵ *denotes the empty content model, pcdata denotes the text content,* $f \in \Sigma'_E$, "," *and "|" stand for concatenation and union (of content models* α_1, α_2, ..., α_n*), and "*", "+", and "?" stand for zero or more, one or more, and optional occurrence(s) (of content model* β*) respectively.*

Definition 3. An XML schema S *is a four-tuple* $(\Sigma'_E, \Sigma'_A, \Delta, s)$, *where*

- Σ'_E *is a finite set of element names,*
- Σ'_A *is a finite set of attribute names,*
- Δ *is a finite set of declarations of the form* $e \rightarrow \alpha$ *or* $e \rightarrow \beta$, *where* $e \in \Sigma'_E$, α *is a content model over* Σ'_E, *and* $\beta \subseteq \Sigma'_A$, *and*
- $s \in \Sigma'_E$ *is a start symbol.*

Definition 4. *An XML document* $T = (V, E, \Sigma_E, \Sigma_A, \Gamma, lab, r)$ *is* valid against XML schema $S = (\Sigma'_E, \Sigma'_A, \Delta, s)$ *if* $lab(r) = s$ *and for* $\forall\, v \in V$ *s.t.* $lab(v) \in \Sigma_E$*:*

- $lab(v) \in \Sigma'_E$,
- *the sequence of labels of its child nodes* $\langle e_{i;i=1,...,k}$ *s.t.* $e_i \in \Sigma_E \cup \Gamma \rangle$ *matches the content model* α, *where* $(lab(v) \rightarrow \alpha) \in \Delta$, *and*
- *the set of labels of its child nodes* $\{a_{i;i=1,...,l}$ *s.t.* $a_i \in \Sigma_A\} \subseteq \beta$, *where* $(lab(v) \rightarrow \beta) \in \Delta$.

To simplify the XML-to-relational mapping process an XML schema is often transformed into a graph representation. Probably the first occurrence of this representation, so-called *DTD graph*, can be found in [29]. There are also various other types of graph representation of an XML schema, nevertheless the analyzed techniques use the following or a similar one. If necessary, we mention the slight differences later in the text.

Definition 5. A schema graph *of a schema* $S = (\Sigma'_E, \Sigma'_A, \Delta, s)$ *is a directed, labeled graph* $G = (V, E, lab')$, *where*

- V *is a finite set of nodes,*
- $E \subseteq V \times V$ *is a set of edges,*
- $lab' : V \rightarrow \Sigma'_E \cup \Sigma'_A \cup \{$ "|", "*", "+", "?", ","$\} \cup \{pcdata\}$ *is a surjective function which assigns a label to* $\forall\, v \in V$, *and*
- s *is the root node of the graph.*

The core idea of XML-to-relational mapping methods is to *decompose* a given schema graph into *fragments*, which are mapped to corresponding relations.

Definition 6. A fragment f *of a schema graph* G *is each its connected subgraph.*

Definition 7. A decomposition *of a schema graph* G *is a set of its fragments* $\{f_1, ..., f_n\}$, *where* $\forall\ v \in V$ *is a member of at least one fragment.*

3 Existing Approaches and Their Classification

Up to now only a few papers have focused on a proposal of a database-based XML-processing method which is able to adapt the target database schema to given information or typical situations. We distinguish two main directions – *cost-driven* and *user-driven*. (We wittingly use the term "user-driven" to distinguish the approach from the "user-defined" one.)

Techniques of the former group can choose the most efficient XML-to-relational storage strategy automatically. They usually evaluate a subset of possible mappings and choose the best one according to the given sample of XML data, query workload, etc. The main advantage is expressed by the adverb "automatically", i.e. without necessary or undesirable user interference.

By contrast, techniques of the latter group also support several storage strategies but the final decision is left in hands of users. We distinguish these techniques from the user-defined ones since their approach is slightly different: By default they offer a fixed mapping. But users can influence the mapping process by annotating fragments of the input XML schema (which should not be mapped by default) with demanded storage strategies. In other words the user is enabled to improve the fixed strategy. Similarly to the user-defined techniques this approach also assumes a skilled user, but, on the other hand, most of the work is done by the system itself and the user is expected to help the mapping process, not to perform it.

In the following subsections we briefly describe the found existing methods of the two approaches and we further classify their features.

3.1 Cost-Driven Techniques

As mentioned above, cost-driven techniques can choose the best storage strategy for a particular application automatically, without any interference of a user. In other words the user can influence the mapping process only through the provided XML schema, set of sample XML documents or data statistics, set of XML queries and eventually their weights, etc.

Each of the techniques can be characterized by following five features:

- an initial XML schema S_{init},
- a set of XML schema transformations $T = \{t_1, t_2, ..., t_n\}$, where $\forall\ i : t_i$ transforms a given schema S_1 into a schema S_2,
- a fixed XML-to-relational mapping function f_{map} which transforms a given XML schema S into a relational schema R,
- a set of input sample data D_{sample} which characterizes the future application, and

– a cost function f_{cost} which evaluates the efficiency of the given relational schema R with regard to the set D_{sample}.

The required result is an optimal relational schema R_{opt}, i.e. a schema, where $f_{cost}(R_{opt}, D_{sample})$ is minimal.

It is important to mention that though the set of transformations T is always finite, they often generate a possibly infinite set of XML schemes. It is caused by the fact that particular transformations usually have various parameters or can be applied to any subgraph of schema S_1.

A naive, but illustrative, cost-driven storage strategy that is based on the idea of using a "brute force" is depicted by Algorithm 1.

Algorithm 1 A naive search algorithm

Input: S_{init}, T, f_{map}, D_{sample}, f_{cost}
Output: R_{opt}
1: $S \leftarrow \{S_{init}\}$
2: **while** $\exists\ t \in T, s \in S : t(s) \notin S$ **do**
3: $\quad S \leftarrow S \cup \{t(s)\}$
4: **end while**
5: $cost_{opt} \leftarrow \infty$
6: **for all** $s \in S$ **do**
7: $\quad R_{tmp} \leftarrow f_{map}(s)$
8: $\quad cost_{tmp} \leftarrow f_{cost}(R_{tmp}, D_{sample})$
9: $\quad$ **if** $cost_{tmp} < cost_{opt}$ **then**
10: $\quad\quad R_{opt} \leftarrow R_{tmp}$
11: $\quad\quad cost_{opt} \leftarrow cost_{tmp}$
12: $\quad$ **end if**
13: **end for**
14: **return** R_{opt}

The naive algorithm first generates a set of possible XML schemes S using transformations from set T and starting from initial schema S_{init} (lines 1 – 4). Then it searches for schema $s \in S$ with minimal cost $f_{cost}(f_{map}(s), D_{sample})$ (lines 5 – 13) and returns the corresponding optimal relational schema $R_{opt} = f_{map}(s)$. It is obvious that the complexity of such algorithm strongly depends on the set T. It can be proven that even a simple set of transformations causes the problem of finding the optimal schema to be NP-hard [32] [34] [18]. Thus the techniques in fact search for a suboptimal solution using various heuristics, greedy strategies, approximation algorithms, terminal conditions, etc.

We can also observe that purely fixed methods can be considered as a special type of cost-driven methods, where $T = \emptyset$, $D_{sample} = \emptyset$, and $f_{cost}(R, \emptyset) = const$ for $\forall\ R$.

Hybrid Object-Relational Mapping One of the first attempts of a cost-driven adaptive approach is a method called *Hybrid object-relational mapping* [14] [15]. It is based on the fact that if XML documents are mostly semi-structured, a "classical" decomposition of unstructured or semi-structured XML parts into relations (e.g. [29]) leads to inefficient query processing caused by plenty of inevitable join operations. The algorithm exploits the idea of storing well structured parts into relations and semi-structured parts in a more natural way – using so-called *XML data type*, which supports path queries and XML-aware full-text operations. The fixed mapping for structured parts is similar to the classical *Hybrid algorithm* [29], whereas in addition it exploits NF^2-relations using constructs such as `set-of`, `tuple-of`, and `list-of`.

The main concern of the method is to identify the structured and semi-structured parts. The process consist of the following steps:

1. A schema graph $G_1 = (V_1, E_1, lab'_1)$ is built for a given DTD.
2. For $\forall\ v \in V_1$ a *measure of significance* ω_v (see below) is determined.
3. Each $v \in V_1$ which satisfies the following conditions is identified:
   (a) v is not a leaf node.
   (b) For v and $\forall$ its descendant $v_{i;1 \leq i \leq k} : \omega_v < \omega_{LOD}$ and $\omega_{v_i} < \omega_{LOD}$, where ω_{LOD} is a required *level of detail* of the resulting schema.
   (c) v does not have a parent node which would satisfy the conditions too.
4. Each fragment $f \subseteq G_1$ which consists of a previously identified node v and its descendants is replaced with an attribute node having the XML data type, resulting in a schema graph G_2.
5. G_2 is mapped to a relational schema using a fixed mapping.

The measure of significance ω_v of a node v is defined as

$$\omega_v = \frac{1}{2}\omega_{S_v} + \frac{1}{4}\omega_{D_v} + \frac{1}{4}\omega_{Q_v} \tag{1}$$

$$\omega_{D_v} = \frac{card(D_v)}{card(D)} \tag{2}$$

$$\omega_{Q_v} = \frac{card(Q_v)}{card(Q)} \tag{3}$$

where

- ω_{S_v} is derived from the DTD structure as a combination of weights expressing position of v in the graph and complexity of its content model[3],
- $D \subseteq D_{sample}$ is a set of all given documents,
- $D_v \subseteq D$ is a set of documents containing v,
- $Q \subseteq D_{sample}$ is a set of all given queries, and
- $Q_v \subseteq Q$ is a set of queries containing v.

[3] For more details see [14] or [15].

As we can see, the algorithm optimizes the naive approach mainly in the following points:

- The schema graph is preprocessed, i.e. ω_v is determined for $\forall\ v \in V_1$.
- The set of transformations T is a singleton.
- The transformation is performed if the current node satisfies the above mentioned conditions a) – c).

As it is obvious, the preprocessing ensures that the complexity of the search algorithm is given by $K_1 * card(V_1) + K_2 * card(E_1)$, where $K_1, K_2 \in N$. On the other hand, the optimization is too restrictive in terms of the amount of possible XML-to-relational mappings.

LegoDB Mapping Another example of adaptive cost-driven methods, was proposed and implemented for *LegoDB system* [9] [10] and later enhanced and extended into *FlexMap framework* [27] [28]. The algorithm optimizes the naive approach using a simple greedy strategy as depicted in Algorithm 2.

Algorithm 2 A greedy search algorithm

Input: S_{init}, T, f_{map}, D_{sample}, f_{cost}
Output: R_{opt}

```
 1: S_opt ← S_init
 2: R_opt ← f_map(S_opt)
 3: cost_opt ← f_cost(R_opt, D_sample)
 4: loop
 5:    cost_min ← ∞
 6:    for all t ∈ T do
 7:       cost_t ← f_cost(f_map(t(S_opt)), D_sample)
 8:       if cost_t < cost_min then
 9:          t_min ← t
10:          cost_min ← cost_t
11:       end if
12:    end for
13:    if cost_min < cost_opt then
14:       S_opt ← t_min(S_opt)
15:       R_opt ← f_map(S_opt)
16:       cost_opt ← f_cost(R_opt, D_sample)
17:    else
18:       break;
19:    end if
20: end loop
21: return R_opt
```

The main differences in comparison with the naive approach are the choice of the least expensive transformation at each iteration (lines 5 – 12) and the

termination of searching if there exists no transformation $t \in T$ that can reduce the current (sub)optimum (lines 13 – 19).

The set T of XML-to-XML transformations involves following XSD modifications:

- *Inlining and outlining* – mutually inverse operations which enable to store columns of a subelement or attribute either in a parent table or in a separate table
- *Splitting and merging elements* – mutually inverse operations which enable to store columns of a shared element[4] either in a common table or in separate tables, each for a particular sharer
- *Associativity and commutativity* – operations which enable to group different elements into one table
- *Union distribution and factorization* – mutually inverse operations which enable to separate out components of a union using equation $(a, (b|c)) = ((a, b)|(a, c))$
- *Splitting and merging repetitions* – exploitation of equation $(a+) = (a, a*)$
- *Simplifying unions* – exploitation of equation $(a|b) \subseteq (a?, b?)$

Note that except for commutativity and simplifying unions the transformations generate equivalent schema in terms of equivalence of sets of document instances. Commutativity does not retain the order of the schema, while simplifying unions generates a more general schema, i.e. a schema with larger set of allowed document instances. (Unfortunately only a subset of the mentioned transformations – namely inlining and outlining – was implemented and experimentally tested by the FlexMap system.)

The fixed mapping again uses a strategy similar to the Hybrid algorithm but it is applied locally on each fragment of the schema. The fragments are specified by the transformation rules stated by the search algorithm. For example elements determined to be outlined are not inlined though a "traditional" Hybrid algorithm would do so.

The process of evaluating f_{cost} is significantly optimized. A naive approach would require:

1. construction of a particular relational schema,
2. loading sample XML data into the relations, and
3. cost analysis of the resulting relational structures.

The LegoDB evaluation exploits an XML Schema-aware statistics framework *StatiX* [13] which analyzes the structure of a given XSD and XML documents and computes their statistical summary. The XML statistics are then "mapped" to relational statistics regarding the fixed XML-to-relational mapping and together with sample query workload used as an input for a classical relational optimizer which estimates the resulting cost. Thus no relational schema has to be constructed.

[4] An element with multiple parent elements in the schema – see [29].

Furthermore, as the statistics are respectively updated at each XML-to-XML transformation, the XML documents need to be processed only once.

An Adjustable and Adaptable Method (AAM) The following method, which is also based on the idea of searching a space of possible mappings, is presented in [32] as an *Adjustable and adaptable method (AAM)*. In this case the authors adapt the given problem to features of genetic algorithms. This is also the first paper that mentions that the problem of finding a relational schema R for a given set of XML documents and queries D_{sample}, s.t. $f_{cost}(R, D_{sample})$ is minimal, is NP-hard in the size of the data.

The set T of XML-to-XML transformations consists of inlining and outlining of subelements. For the purpose of the genetic algorithm each transformed schema is represented using a bit string, where each bit corresponds to an edge of the schema graph and it is set to

- 1 if the element the edge points to is stored into a separate table, or
- 0 if the element the edge points to is stored into parent table.

The bits set to 1 represent "borders" among fragments, whereas each fragment is stored into one table corresponding to so-called *Universal table* [12]. The extreme instances correspond to "one table for the whole schema" (in case of 00...0 bit string) resulting in many null values and "one table per each element" (in case of 11...1 bit string) resulting in many join operations.

Similarly to the previous strategy the genetic algorithm chooses only the best possible continuation at each iteration. The algorithm consists of the following steps:

1. The initial population P_0 (i.e. the set of schema bit strings) is generated randomly.
2. The following steps are repeated until terminating conditions are met:
   (a) Each member of the current population P_i is evaluated and only the best representatives are selected for further production.
   (b) The next generation P_{i+1} is produced by genetic operators *crossover*, *mutation*, and *propagate*.

The algorithm terminates either after certain number of transformations or if a good-enough schema is achieved.

The cost function $f_{cost}(R, D_{sample})$ is expressed as:

$$f_{cost}(R, D_{sample}) = f_M(R, D_{sample}) + f_Q(R, D_{sample}) \tag{4}$$

$$f_M(R, D_{sample}) = \sum_{l=1}^{q} C_l * R_l \tag{5}$$

$$f_Q(R, D_{sample}) = \sum_{i=1}^{m} S_i * P_{S_i} + \sum_{k=1}^{n} J_k * P_{J_k} \quad (6)$$

where

- $f_M(R, D_{sample})$ is a *space-cost function*, where C_l is number of columns and R_l is number of rows in table T_l created for l-th element in the schema,
- q is the number of all elements in the schema,
- $f_Q(R, D_{sample})$ is a *query-cost function*, where S_i is cost and P_{S_i} is probability of i-th given select query and J_k is cost and P_{J_k} is probability of k-th given join query,
- m is the number of select queries in D_{sample}, and
- n is the number of join queries in D_{sample}.

In other words $f_M(R, D_{sample})$ represents the total memory cost of the mapping instance, whereas $f_Q(R, D_{sample})$ represents the total query cost. The probabilities P_{S_i} and P_{J_k} enable to specify which elements will (not) be often retrieved and which sets of elements will (not) be often combined to search. Note that the cost function does not involve a sample set of XML documents at all.

As we can see, this algorithm represents another way of finding a reasonable suboptimal solution in the theoretically infinite set of possibilities – using (in this case two) terminal conditions.

A Hill Climbing Algorithm The last but not least cost-driven adaptable representative can be found in paper [34]. The approach is again based on a greedy type of algorithm, in this case a *Hill climbing strategy* that is depicted by Algorithm 3.

As we can see, the hill climbing strategy differs from the simple greedy strategy depicted in Algorithm 2 in the way it chooses the appropriate transformation $t \in T$. In the previous case the least expensive transformation that can reduce the current (sub)optimum is chosen, in this case it is the first such transformation found.

The schema transformations are based on the idea of vertical (V) or horizontal (H) cutting and merging the given XML schema fragment(s). The set T consists of the following four types of (pairwise inverse) operations:

- *V-Cut(f, (u,v))* – cuts fragment f into fragments f_1 and f_2, s.t. $f_1 \cup f_2 = f$, where (u, v) is an edge from f_1 to f_2, i.e. $u \in f_1$ and $v \in f_2$
- *V-Merge*(f_1, f_2) – merges fragments f_1 and f_2 into fragment $f = f_1 \cup f_2$
- *H-Cut(f, (u,v))* – splits fragment f into twin fragments f_1 and f_2 horizontally from edge (u, v), where $u \notin f$ and $v \in f$, s.t. $ext(f_1) \cup ext(f_2) = ext(f)$ and $ext(f_1) \cap ext(f_2) = \emptyset$ [5] [6]

[5] $ext(f_i)$ is the set of all document-instance fragments conforming to the schema fragment f_i.

[6] Fragments f_1 and f_2 are called *twins* if $ext(f_1) \cap ext(f_2) = \emptyset$ and for each node $u \in f_1$, there is a node $v \in f_2$ with the same label and vice versa.

Algorithm 3 A hill climbing algorithm

Input: S_{init}, T, f_{map}, D_{sample}, f_{cost}
Output: R_{opt}

```
1: S_opt ← S_init
2: R_opt ← f_map(S_opt)
3: cost_opt ← f_cost(R_opt, D_sample)
4: T_tmp ← T
5: while T_tmp ≠ ∅ do
6:    t ← any member of T_tmp
7:    T_tmp ← T_tmp\{t}
8:    S_tmp ← t(S_opt)
9:    cost_tmp ← f_cost(f_map(S_tmp), D_sample)
10:   if cost_tmp < cost_opt then
11:      S_opt ← S_tmp
12:      R_opt ← f_map(S_tmp)
13:      cost_opt ← cost_tmp
14:      T_tmp ← T
15:   end if
16: end while
17: return R_opt
```

– *H-Merge*(f_1, f_2) – merges two twin fragments f_1 and f_2 into one fragment f s.t. $ext(f_1) \cup ext(f_2) = ext(f)$

As we can observe, *V-Cut* and *V-Merge* operations are similar to outlining and inlining of the fragment f_2 out of or into the fragment f_1. On the other hand, *H-Cut* operation, which is in practice applied only on shared fragments, corresponds to splitting of elements mentioned in LegoDB mapping, i.e. duplication of the shared part. Likewise the *H-Merge* operation corresponds to inverse merging of elements.

The fixed XML-to-relational mapping maps each fragment f_i which consists of nodes $\{v_1, v_2, ..., v_n\}$ to relation

$R_i = (id(r_i) : int, id(r_i.parent) : int, lab(v_1) : type(v_1), ..., lab(v_n) : type(v_n))$ where r_i is root element of f_i. Note that such mapping is again similar to locally applied Universal table.

The cost function $f_{cost}(R, D_{sample})$ is expressed as

$$f_{cost}(R, D_{sample}) = \sum_{i=1}^{n} w_i * cost(Q_i, R) \tag{7}$$

where D_{sample} consists of a sample set of XML documents and a given query workload $\{(Q_i, w_i)_{i=1,2,...,n}\}$, where Q_i is an XML query and w_i is its weight. The cost function $cost(Q_i, R)$ for a query Q_i which accesses fragment set $\{f_{i1}, ..., f_{im}\}$ is expressed as

$$cost(Q_i, R) = \begin{cases} |f_{i1}| & m = 1 \\ \sum_{j,k} (|f_{ij}| * Sel_{ij} + \delta * (|E_{ij}| + |E_{ik}|)/2) & m > 1 \end{cases} \tag{8}$$

where f_{ij} and f_{ik}, $j \neq k$ are two join fragments, $|E_{ij}|$ is the number of elements in $ext(f_{ij})$, and Sel_{ij} is the selectivity of the path from the root to f_{ij} estimated using *Markov table*. In other words the formula simulates the cost for joining relations corresponding to fragments f_{ij} and f_{ik}.

The authors further analyze the influence of the choice of initial schema S_{init} on efficiency of the search algorithm. They analyze three types of initial schema decompositions leading to Attribute [12], Shared, or Hybrid [29] mapping. The paper concludes with the finding that a good choice of an initial schema is crucial and can lead to faster searches of the suboptimal mapping.

3.2 User-Driven Techniques

As mentioned above, the most flexible approach is "to leave the whole process in hands of a user" who defines both the target database schema and the required mapping. We speak about so-called *user-defined mapping* techniques. Probably due to simple implementation they are especially popular and supported in most commercial database systems[7].

At first sight the idea is correct – users can decide what suits them most and are not restricted by features and especially disadvantages of a particular technique. The problem is that such approach assumes users skilled in two complex technologies – (object-)relational databases and XML. Furthermore, for more complex applications the design of an optimal relational schema is generally not an easy task.

On this account several new techniques – for the purpose of this paper called *user-driven* mapping strategies – were proposed. The main difference is that the user can influence a default fixed mapping strategy using annotations which specify the required mapping for particular schema fragments. The set of allowed mappings is naturally limited but still enough powerful to define various mapping strategies.

Each of the techniques is characterized by following four features:

- an initial XML schema S_{init},
- a set of allowed fixed XML-to-relational mappings $\{f^i_{map}\}_{i=1,\ldots,n}$,
- a set of annotations A, each of which is specified by name, target, allowed values, and function, and
- a default mapping strategy f_{def} for not annotated fragments.

MDF Probably the first approach which faces the mentioned issues is proposed in papers [4] [3] (which extend ideas of papers [2] [5]) as a *Mapping definition framework (MDF)*. It allows users to specify the required mapping and it is able to check *correctness* and *completeness* of such specifications and to complete possible incompleteness. The mapping specifications are made by annotating the input XSD with a predefined set of annotations, i.e. attributes from namespace called `mdf`. The set of annotating attributes A is listed in Table 1.

[7] An overview and analysis of commercial user-defined approaches can be found in [2].

Attribute	Target	Value	Function
`outline`	attribute or element	`true`, `false`	If the value is `true`, a separate table is created for the attribute / element. Otherwise it is inlined to parent table.
`tablename`	attribute, element, or group	`string`	The string is used as the table name.
`columnname`	attribute, element, or simple type	`string`	The string is used as the column name.
`sqltype`	attribute, element, or simple type	`string`	The string defines the SQL type of a column.
`structurescheme`	root element	`KFO`, `Interval`, `Dewey`	Defines the way of capturing the structure of the whole schema.
`edgemapping`	element	`true`, `false`	If the value is `true`, the element and all its subelements are mapped using Edge mapping.
`maptoclob`	attribute or element	`true`, `false`	If the value is `true`, the element / attribute is mapped to a CLOB column.

Table 1. Annotation attributes for MDF

As we can see from the table, the set of allowed XML-to-relational mappings $\{f^i_{map}\}_{i=1,\ldots,n}$ involves inlining and outlining of an element or attribute, *Edge mapping* [12] strategy, and mapping an element or attribute to a CLOB column. Furthermore, it enables to specify the required capturing of the structure of the whole schema using one of the following three approaches:

- *Key, Foreign Key and Ordinal Strategy (KFO)* – each node is assigned a unique ID and a foreign key pointing to parent ID, the sibling order is captured using an ordinal value
- *Interval Encoding* – a unique {`start,end`} interval is assigned to each node corresponding to preorder and postorder traversal entering time
- *Dewey Decimal Classification* – each node is assigned a path to the root node described using concatenation of node IDs along the path

As side effects can be considered attributes for specifying names of tables or columns and data types of columns. Not annotated parts are stored using user-predefined rules, whereas such mapping is always a fixed one.

XCacheDB System Paper [6] also proposes a user-driven mapping strategy which is implemented and experimentally tested as an *XCacheDB system*. Similarly to the previous case a user can provide an annotated XML schema which

Attribute	Value	Function
`INLINE`	$\emptyset$	If placed on a node v, the fragment rooted at v is inlined into parent table.
`TABLE`	$\emptyset$	If placed on a node v, a new table is created for the fragment rooted at v.
`STORE_BLOB`	$\emptyset$	If placed on a node v, the fragment rooted at v is stored also into a BLOB column.
`BLOB_ONLY`	$\emptyset$	If placed on a node v, the fragment rooted at v is stored into a BLOB column.
`RENAME`	`string`	The value specifies the name of corresponding table or column created for node v.
`DATATYPE`	`string`	The value specifies the data type of corresponding column created for node v.

Table 2. Annotation attributes for XCacheDB

contains the demanded mappings for particular schema fragments, otherwise a default strategy is used. Unfortunately the system considers only unordered and acyclic XML schemes and omits mixed-content elements.

The set of annotating attributes A that can be assigned to any node $v \in S_{init}$ is listed in Table 2. As we can see, it enables inlining and outlining of a node, storing a fragment into a BLOB column, specifying table names or column names, and specifying column data types. The main difference is in the data redundancy allowed by attribute `STORE_BLOB` which enables to shred the data into table(s) and at the same time to store pre-parsed XML fragments into a BLOB column.

The fixed mapping uses a slightly different strategy: Each element or attribute node is assigned a unique ID. Each fragment f is mapped to a table T_f which has an attribute $a_{v_{ID}}$ of ID data type for each element or attribute node $v \in f$. If v has is an atomic node[8], T_f has also an attribute a_v of the same data type as v. For each distinct path that leads to f from a repeatable ancestor v, T_f has a parent reference column of type ID which points to ID of v. Note that this mapping strategy is again a fixed one.

For better lucidity we recapitulate the main features of the mentioned cost-driven and user-driven approaches in Tables 3 and 4 respectively.

3.3 Theoretic Issues

Besides proposals of cost-driven and user-driven techniques there are also papers which discuss the corresponding open issues of various XML-to-relational mappings and their efficiency on theoretic level.

[8] An attribute node or an element node having no subelements.

Method	S_{init}	T	f_{map}	D_{sample}	f_{cost}
Hybrid OR	user-given DTD	Semi-structured fragments are replaced with an attribute having an XML data type	Hybrid algorithm modified for NF^2-relations	schema, documents, queries	The measure of significance ω_v of node v must be below the level of detail ω_{LOD}
LegoDB	user-given XSD	inlining / outlining, splitting / merging elements, associativity, commutativity, union distribution / factorization, splitting / merging repetitions, simplifying unions	Hybrid algorithm applied locally on each fragment	schema, documents, queries	Performed by relational optimizer with input based on XML data statistics
AAM	randomly generated set of schema decompositions	inlining / outlining	Each fragment is mapped to one table similar to the Universal table	schema, queries + probabilities	The total memory cost $f_M(R, D_{sample})$ + the total query cost $f_Q(R, D_{sample})$
Hill Climbing	decomposed user-given DTD leading to Attribute, Shared, or Hybrid mapping	V-Cut / V-Merge, H-Cut / H-Merge	Each fragment is mapped to one table similar to the Universal table	schema, documents, queries + weights	$\sum_{i=1}^{n} w_i * cost(Q_i, R)$, where $cost(Q_i, R)$ is the cost estimation of query Q_i and w_i is its weight

Table 3. Overview of characteristics of cost-driven methods

Method	S_{init}	$\{f^i_{map}\}_{i=1,\ldots,n}$	A	f_{def}
MDF	user-given XSD	inlining / outlining, BLOB, Edge mapping + capturing of the structure using Key, Foreign Key and Ordinal Strategy / Interval Encoding / Dewey Decimal Classification	`outline`, `tablename`, `columnname`, `sqltype`, `structurescheme`, `edgemapping`, `maptoclob`	user-predefined rules
XCacheDB	user-given XSD	inlining / outlining, BLOB	`INLINE`, `TABLE`, `STORE_BLOB`, `BLOB_ONLY`, `RENAME`, `DATATYPE`	Each fragment is mapped to one table with an ID attribute for each element / attribute, data type attribute for each atomic node, and foreign key to each repeatable ancestor

Table 4. Overview of characteristics of user-driven methods

Data Redundancy As mentioned above, the XCacheDB system allows a certain degree of redundancy to ensure more efficient query processing. The corresponding paper [6] discusses the strategy also on theoretic level. There are two main representatives of the allowed redundancy – BLOB columns and the violation of BCNF or 3NF condition. On this account the authors define four classes of XML schema decompositions.

Before we state the definitions we have to note that this approach is based on a slightly different graph representation of a schema than was defined by Definition 5. In this case nodes of the graph correspond to elements, attributes, or pcdata, while edges are labeled with corresponding operators.

Definition 8. *A schema* decomposition is minimal *if all edges connecting nodes of different fragments are labeled with "*" or "+".*

Definition 9. *A schema* decomposition is 4NF *if all fragments are 4NF fragments. A* fragment is 4NF *if no two nodes of the fragment are connected by a "*" or "+" labeled edge.*

Definition 10. *A schema* decomposition is non-MVD *if all fragments are non-MVD fragments. A* fragment is non-MVD *if all "*" or "+" labeled edges appear in a single path.*

Definition 11. *A schema* decomposition is inlined *if it is non-MVD but it is not a 4NF decomposition. A* fragment is inlined *it is non-MVD but it is not a 4NF fragment.*

According to these definitions fixed mapping strategies (e.g. [29] [20]) naturally consider only 4NF decompositions which are least space-consuming and seem to be the best choice if we do not consider any other information. Paper [6] shows that having further information (in this particular case given by a user), the choice of other type of decomposition can lead to more efficient query processing though it requires a certain level of redundancy.

Grouping problem On the other hand, paper [18] is dealing with the idea that searching a (sub)optimal relational decomposition is not only related to given XML schema, query workload, and XML data, but it is also highly influenced by the chosen *query translation algorithm*[9] and the cost model.

For the theoretic purpose a subset of the problem – so-called *grouping problem* – is considered. It deals with possible storage strategies for shared subelements, i.e. either into one common table (so-called *fully grouped strategy*) or into separate tables, one for each sharer (so-called *fully partitioned strategy*). For analysis of its complexity the authors further define two simple cost metrics:

- *RelCount* – the cost of a relational query is the number of relation instances in the relational algebra expression
- *RelSize* – the cost of a relational query is the sum of the number of tuples in relation instances in the relational algebra expression

and three query translation algorithms:

- *Naive Translation* – performs a join between the relations corresponding to all the elements appearing in the query, a *wild-card query*[10] is converted into union of several queries, one for each satisfying wild-card substitution
- *Single Scan* – a separate relational query is issued for each leaf element and joins all relations on the path until the least common ancestor of all the leaf elements is reached
- *Multiple Scan* – on each relation containing a part of the result is applied Single Scan algorithm and the resulting query consists of union of the partial queries

On a simple example the authors show that for a wild-card query Q which retrieves a shared fragment f with algorithm Naive Translation the fully partitioned strategy performs better, whereas with algorithm Multiple Scan the fully grouped strategy performs better. Furthermore, they illustrate that reliability of the chosen cost model is also closely related to query translation strategy. If a query contains not very selective predicate than the optimizer may choose a plan that scans corresponding relations and thus RelSize is a good corresponding metric. On the other hand, in case of highly selective predicate the optimizer may choose an index lookup plan and thus RelCount is a good metric.

[9] An algorithm for translating XML queries into SQL queries
[10] A query containing "`//`" or "`/*`" operators.

Last but not least the authors theoretically prove that various combinations of the above mentioned cost metrics and translation algorithms can produce differently complex problems, up to NP-hard ones.

4 Summary

We can sum up the state of the art of adaptability of database-based XML-processing methods into following natural but important findings:

1. As the storage strategy has a crucial impact on query-processing performance, a fixed mapping based on predefined rules and heuristics is not universally efficient.
2. It is not an easy task to choose an optimal mapping strategy for a particular application and thus it is not advisable to rely only on user's experience and intuition.
3. As the space of possible XML-to-relational mappings is very large (usually theoretically infinite) and most of the subproblems are even NP-hard, the exhaustive search is impractical and often even impossible. It is necessary to define search heuristics, approximation algorithms, and/or reliable terminal conditions.
4. The choice of an initial schema can strongly influence the efficiency of the search algorithm. It is reasonable to start with at least "locally good" schema.
5. A strategy of finding a (sub)optimal XML schema should take into account not only the given schema, query workload, and XML data statistics, but also consider possible query translations, cost metrics, and their consequences.
6. Cost evaluation of a particular XML-to-relational mapping should not involve time-consuming construction of a particular relational schema, loading sample XML data and analyzing the resulting relational structures. It can be optimized using cost estimation of XML queries, XML data statistics, etc.
7. Despite the previous claim, the user should be allowed to influence the mapping strategy. On the other hand, the approach should not demand a full schema specification but it should be able to efficiently complete the user-given hints.
8. Even thought a storage strategy is able to adapt to a given sample of schemes, data, queries, etc., its efficiency is still endangered by later changes of the expected usage.

5 Open Issues

Although each of the existing approaches brings certain interesting ideas and optimizations, there is still a space of possible future improvements of the adaptable methods. We describe and discuss them in this section starting from (in our opinion) the least complex ones.

5.1 Problem of Missing Input Data

As we already know, the set of input data D_{sample} for cost-driven adaptive methods usually consists of:

- an XML schema S,
- a set of XML documents $\{d_1, d_2, .., d_k\}$ valid against S, and
- a set of XML queries $\{q_1, q_2, .., q_l\}$ over S, eventually with corresponding weights $\{w_1, w_2, .., w_l\}$, $\forall\ i : w_i \in \langle 0, 1 \rangle$.

The problem of missing input XML schema was already outlined in the introduction in connection with advantages and disadvantages of generic and schema-driven methods. As we suppose that the adaptability is the ability to adapt to the given situation, an adaptive method which does not depend on existence of an XML schema but can exploit the information if being given is probably a natural first type of improvement. This idea is also strongly related to the earlier mentioned problem of choice of a locally good initial schema S_{init}. The corresponding main questions are:

- Can be the user-given schema considered as a good candidate for initial schema S_{init}?
- How can we measure this quality?
- How can we (efficiently) find an eventual better candidate?
- Can we find such candidate for schema-less XML documents?

A possible solution can be found in exploitation of methods for automatic construction of XML schema for the given set of XML documents (e.g. [23] [25]). These methods are able to derive corresponding content models from a given sample set of (similar) XML documents. Thus if we assume that documents are more precise sources of structural information, we can expect that a schema generated on their bases will have good characteristics.

On the other hand, the problem of missing input XML documents can be at least partly solved using reasonable default settings based on general analysis of real XML data (e.g. [19] [22]). Furthermore, the surveys show that real XML data is surprisingly simple thus the default mapping strategy does not have to be complex too. It should rather focus on efficient processing of frequently used XML patterns.

On the other hand, the presence of sample query workload is crucial since (to our knowledge) there are no analyses on real XML queries, i.e. no source of information for default settings. The reason is that the way how to collect such real representatives is not as straightforward as in case of XML documents, which can be easily crawled from the Internet. The best source of XML queries are currently XML benchmarking projects (e.g. [26] [33]) but as the data and especially queries are supposed to be used for rating the performance of a system in various situations, they cannot be considered as an example of a real workload.

Naturally the query statistics can be gathered by the system itself and the relational schema can be adapted continuously. But this is already the problem of dynamic adaptability discussed later in section 5.5.

5.2 Efficient Solution of Subproblems

A surprising fact we have encountered are numerous simplifications of the chosen solutions. At it was mentioned, some of the techniques omit e.g. ordering of elements, mixed contents, or recursion. This is a bit confusing finding regarding the fact that there are proposals of efficient processing of these XML constructs (e.g. [30]) and that adaptive methods should be able to cope with various situations.

A similar observation can be done for user-driven methods. Though the proposed systems are able to store schema fragments in various ways, the default mapping strategy for not annotated parts of a given schema is again a fixed one. It seems to be an interesting optimization to join the ideas of cost-driven and user-driven approaches and to search the (sub)optimal mapping for not annotated parts using a cost-driven method.

5.3 Deeper Exploitation of User-Given Information

Another open issue is the problem of possible deeper exploitation of the information given by the user. We can identify two main questions:

1. How can be the user-given information better exploited?
2. Are there any other information a user can provide to increase the efficiency?

A possible answer at least for the first question can be found in the idea of pattern matching. The idea is to use user-given schema annotations as "hints" how to store particular XML patterns which can be further exploited in searching an efficient mapping for not annotated parts. We can naturally predict that structurally similar fragments should be stored similarly and thus to focus on finding these fragments in the rest of the schema. The main problem of this idea is how to identify the structurally similar fragments. If we consider the variety of XML-to-XML transformations, two structurally same fragments can be expressed using "at first glance" different regular expressions. Thus it is necessary to propose particular levels of equality of XML schema fragments and algorithms how to determine them.

5.4 Theoretical Analysis of the Problem

As we can see from the overview of the existing methods, there are various types of XML-to-XML transformations, while the mentioned ones certainly do not cover the whole set of possibilities. Unfortunately there seems to be no theoretic study of these transformations, their key characteristics, and possible classifications. The study can, among others, focus on equivalent and generalizing transformations and as such serve as a good basis for the pattern matching strategy.

Especially interesting will be the question of NP-hardness in connection with the set of allowed transformations and its complexity (similarly to paper [18] which analyzes theoretical complexity of combinations of cost metrics and query translation algorithms). Such survey will provide useful information especially for optimizations of the search algorithm.

5.5 Dynamic Adaptability

The last but not least mentioned open issue is naturally connected with the most striking disadvantage of adaptive methods – the problem of possible changes of both XML queries and XML data that can lead to crucial worsening of their efficiency. As it was already mentioned, it is also related to the problem of missing input XML queries and ways how to gather them. Furthermore, the question of changes of XML data opens another wide research area of updatability of the stored data – a feature that is often omitted in current approaches although its importance is crucial.

The solution to these issues – i.e. a system that is able to adapt dynamically – is obvious and challenging but it is not an easy task. It should especially avoid total reconstructions of the whole relational schema and corresponding necessary reinserting of all the stored data, or such operation should be done only in very special cases and not often.

On the other hand, this "brute-force" approach can serve as a good inspiration. It is possible to suppose that changes especially in case of XML queries will not be radical but will have a gradual progress. Thus the changes of the relational schema will be mostly local and we can apply the expensive reconstruction just locally. Furthermore, we can again exploit the idea of pattern matching and try to find the XML pattern defined by the modified schema fragment in the rest of the schema.

Another question is how often should be the relational schema reconstructed. The natural idea is of course "not too often". But, on the other hand, a research can be done on the idea of performing gradual minor changes. It is probable that such approach will lead to less expensive (in terms of reconstruction) and at the same time more efficient (in terms of query processing) system. The former hypothesis should be verified, the latter one can be almost certainly expected. The key issue is how to find a reasonable compromise.

6 Conclusion

The main goal of this paper was to describe and discuss the current state of the art and open issues of adaptability in database-based XML-processing methods. First of all, we have stated the reasons why this topic should be ever studied. Then we have provided an overview and classification of the existing approaches and their features and summed up the key findings. Finally, we have discussed the corresponding open issues and their possible solutions.

Our aim was to show that the idea of processing XML data using relational databases is still up to date and should be further developed. From the overview of the state of the art we can see that even though there are interesting and inspiring approaches, there is still a variety of open problems which can further improve the database-based XML processing.

Our future work will naturally follow the open issues stated at the end of this paper and especially survey into the possible solutions we have mentioned.

Firstly, we will focus on the idea of improving the user-driven techniques using adaptive algorithm for not annotated parts of the schema together with deeper exploitation of the user-given hints using pattern-matching methods – i.e. a hybrid user-driven cost-based system. Secondly, we will deal with the problem of missing theoretic study of schema transformations, their classification, and particularly influence on the complexity of the search algorithm. Finally, on the basis of the theoretical study and the hybrid system we will study and experimentally analyze the dynamic enhancing of the system.

It is important to mention that though all the open issues can be studied from various points of view, they are still closely related and influence each other. Thus it is always important to consider the given problem globally and do not omit important consequences.

Acknowledgement

This work was supported in part by Czech Science Foundation (GACR), grant number 201/06/0756.

References

1. *Oracle Database 10g*. Oracle Corporation. `http://www.oracle.com/database/`.
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, F. Du, and J. Freire. A Generic and Flexible Framework for Mapping XML Documents into Relations. In *VLDB'04: Proceedings of 30th International Conference on Very Large Data Bases*, Toronto, ON, Canada, 2004. Morgan Kaufmann Publishers Inc.
5. S. Amer-Yahia and D. Srivastava. A Mapping Schema and Interface for XML Stores. In *WIDM '02: Proceedings of the 4th International Workshop on Web Information and Data Management*, pages 23–30, McLean, Virginia, USA, 2002. ACM Press.
6. A. Balmin and Y. Papakonstantinou. Storing and Querying XML Data Using Denormalized Relational Databases. *The VLDB Journal*, 14(1):30–49, 2005.
7. G. J. Bex, F. Neven, and J. Van den Bussche. DTDs versus XML Schema: a Practical Study. In *WebDB'04: Proceedings of the 7th International Workshop on the Web and Databases*, pages 79–84, New York, NY, USA, 2004. ACM Press.
8. P. V. Biron and A. Malhotra. *XML Schema Part 2: Datatypes (Second Edition)*. W3C Recommendation, October 2004. `www.w3.org/TR/xmlschema-2/`.
9. P. Bohannon, J. Freire, P. Roy, and J. Simeon. *From XML Schema to Relations: A Cost-based Approach to XML Storage*. Technical report, Bell Laboratories, 2001.
10. 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*, page 64, Washington, DC, USA, 2002. IEEE Computer Society.

11. T. Bray, J. Paoli, C. M. Sperberg-McQueen, E. Maler, and F. Yergeau. *Extensible Markup Language (XML) 1.0 (Third Edition)*. W3C Recommendation, February 2004. `http://www.w3.org/TR/REC-xml/`.
12. D. Florescu and D. Kossmann. *A Performance Evaluation of Alternative Mapping Schemes for Storing XML Data in a Relational Database*. Technical Report 3684, INRIA, France, March 1999.
13. J. Freire, J. R. Haritsa, M. Ramanath, P. Roy, and J. Simeon. StatiX: Making XML Count. In *ACM SIGMOD 2002: Proceedings of the 21st International Conference on Management of Data*, pages 181–192, Madison, Wisconsin, USA, 2002. ACM.
14. M. Klettke and H. Meyer. *Managing XML Documents in Object-Relational Databases*. Rostocker Informatik Fachberichte, 24, 1999.
15. M. Klettke and H. Meyer. XML and Object-Relational Database Systems – Enhancing Structural Mappings Based on Statistics. In *Lecture Notes in Computer Science*, volume 1997, pages 151–170, 2000.
16. M. Kratky, J. Pokorny, and V. Snasel. Indexing XML Data with UB-trees. In *ADBIS'02: Proceedings of International Conference on the Advances in Databases and Information Systems*, pages 155–164, Bratislava, Slovakia, 2002.
17. M. Kratky, J. Pokorny, and V. Snasel. Implementation of XPath Axes in the Multidimensional Approach to Indexing XML Data. In *Proceedings of Current Trends in Database Technology - EDBT 2004 Workshops*, pages 46–60, Heraklion, Crete, Greece, 2004. Springer.
18. R. Krishnamurthy, V. Chakaravarthy, and J. Naughton. On the Difficulty of Finding Optimal Relational Decompositions for XML Workloads: A Complexity Theoretic Perspective. In *ICDT 2003: Proceedings of the 9th International Conference on Database Theory*, pages 270–284, Siena, Italy, 2003. Springer.
19. L. Mignet, D. Barbosa, and P. Veltri. The XML Web: a First Study. In *WWW'03: Proceedings of the 12th international conference on World Wide Web, Volume 2*, pages 500–510, New York, NY, USA, 2003. ACM Press.
20. I. Mlynkova and J. Pokorny. From XML Schema to Object-Relational Database – an XML Schema-Driven Mapping Algorithm. In *Proceedings of IADIS International Conference WWW/Internet 2004*, pages 115–122, Madrid, Spain, 2004.
21. I. Mlynkova and J. Pokorny. XML in the World of (Object-) Relational Database Systems. In *Proceedings of the XIII. International Conference ISD 2004*, pages 63–76, Vilnius, Lithuania, 2004.
22. I. Mlynkova, K. Toman, and J. Pokorny. Statistical Analysis of Real XML Data Collections. In *Proceedings of the 13th International Conference on Management of Data COMAD 2006 (to appear)*, Delhi, India, December 2006.
23. C.-H. Moh, E.-P. Lim, and W. K. Ng. DTD-Miner: A Tool for Mining DTD from XML Documents. In *WECWIS'00: Proceedings of the 2nd International Workshop on Advanced Issues of E-Commerce and Web-Based Information Systems*, pages 144–151, Milpitas, CA, USA, 2000. IEEE.
24. M. Murata, D. Lee, and M. Mani. Taxonomy of XML Schema Languages using Formal Language Theory. In *Extreme Markup Languages*, Montreal, Canada, 2001.
25. S. Nestorov, S. Abiteboul, and R. Motwani. Extracting Schema from Semistructured Data. In *SIGMOD'98: Proceedings of the ACM International Conference On Management of Data*, pages 295–306, Seattle, Washington, DC, USA, 1998. ACM Press.
26. E. Rahm and T. Bohme. *XMach-1: A Benchmark for XML Data Management*. Database Group Leipzig, 2006. `http://dbs.uni-leipzig.de/en/projekte/XML/XmlBenchmarking.html`.

27. M. Ramanath, J. Freire, J. Haritsa, and P. Roy. Searching for Efficient XML-to-Relational Mappings. In *XSym 2003: Proceedings of the 1st International XML Database Symposium*, volume 2824, pages 19–36, Berlin, Germany, 2003. Springer.
28. M. Ramanath, J. Freire, J. Haritsa, and P. Roy. *Searching for Efficient XML-to-Relational Mappings.* Technical Report TR-2003-01, DSL/SERC, Indian Institute of Science, 2003.
29. 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.
30. I. Tatarinov, S. D. Viglas, K. Beyer, J. Shanmugasundaram, E. Shekita, and C. Zhang. Storing and Querying Ordered XML Using a Relational Database System. In *SIGMOD'02: Proceedings of 21st International Conference on Management of Data*, pages 204–215, Madison, Wisconsin, USA, 2002. ACM Press.
31. 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/`.
32. 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 2002 Workshop EEXTT and CAiSE 2002 Workshop DTWeb*, pages 117–130, London, UK, 2003. Springer-Verlag.
33. B. B. Yao and M. T. Ozsu. *XBench – A Family of Benchmarks for XML DBMSs.* University of Waterloo, School of Computer Science, Database Research Group, 2002. `http://se.uwaterloo.ca/~ddbms/projects/xbench/Workload.html`.
34. S. Zheng, J. Wen, and H. Lu. Cost-Driven Storage Schema Selection for XML. In *DASFAA 2003: Proceedings of the 8th International Conference on Database Systems for Advanced Applications*, pages 337–344, Kyoto, Japan, 2003. IEEE Computer Society.