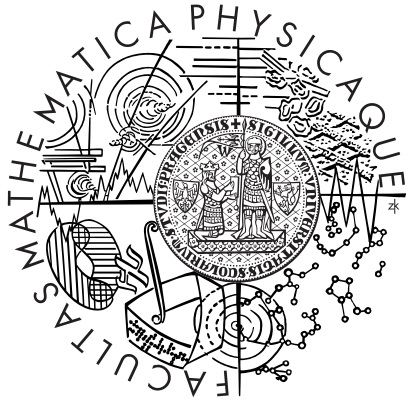
Charles University
Faculty of Mathematics and Physics

DOCTORAL THESIS



RNDr. Irena Mlýnková

XML Data in (Object-)Relational Databases

Department of Software Engineering
Supervisor: *Prof. RNDr. Jaroslav Pokorný, CSc.*

Title: XML Data in (Object-)Relational Databases
Author: RNDr. Irena Mlýnková
Department: Department of Software Engineering
Supervisor: Prof. RNDr. Jaroslav Pokorný, CSc.
Author's e-mail address: irena.mlynkova@mff.cuni.cz
Supervisor's e-mail address: jaroslav.pokorny@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 documents. A natural alternative is to exploit features and functions of (object-)relational database management 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 thesis we focus on further enhancing of current most efficient XML-to-relational storage strategies – so-called adaptive methods. Firstly, we perform a detailed analysis of existing works and especially remaining open issues. Secondly, we propose their enhancing which can be characterized as a hybrid user-driven adaptive mapping strategy, i.e. a combination of so-called user-driven and adaptive methods. In the enhancing we focus especially on deeper exploitation of user-given information, i.e. schema annotations, and we propose an approach which enables to identify new annotation candidates and thus to help users to find a more appropriate mapping strategy. For this purpose we propose a similarity measure which focuses mainly on structural similarity of the given data and an algorithm which enables reasonable tuning of its parameters on the basis of results of statistical analysis of real-world XML data. Using various experiments we show the behavior and efficiency of both the similarity measure and the hybrid mapping strategy on real XML data. And finally, we analyze the correctness and structure of the resulting mapping strategy and related query evaluation. We focus especially on the problem of correction of the set of annotation candidates, evaluation of parts of a single XML query using various storage strategies, and exploitation of redundancy. We conclude with a discussion of further possible improvements of the approach, as well as XML processing using (object-)relational databases in general.

Keywords: XML data management, object-relational databases, XML-to-relational mapping strategy, adaptivity, user-driven method, similarity.

Acknowledgments

I would like to thank all those who supported me in my doctoral studies and work on my thesis. In the first place I very appreciate the help and advices received from my supervisor Jaroslav Pokorný and I am grateful for numerous corrections and comments. Secondly, I would like to thank to Kamil Toman for inspirative cooperation on important part of this work – the statistical analysis of real-world XML data. And undoubtedly, I must also express my thanks to all the anonymous reviewers of my papers for helpful remarks and ideas.

My thanks also go to institutions that provided financial support for my research work. During my doctoral studies, my work was partially supported by the Czech Science Foundation (GAČR), grant number 201/02/1553 and 201/06/0756 and by National Programme of Research, Information Society Project number 1ET100300419.

Last but not least, I am very thankful to René whose unlimited support, patience, and sense of humor made this work possible.

Contents

List of Figures

List of Tables

List of Algorithms

Chapter 1

Introduction

Without any doubt the XML [28] is currently one of the most popular formats for data representation. It is well-defined, easy-to-use, and involves various recommendations such as languages for structural specification, transformation, querying, updating, etc. The popularity invoked an enormous endeavor to propose more efficient methods and tools for managing and processing XML data. 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 management system ((O)RDBMS), methods which exploit a pure object-oriented approach, and native methods that use special indices, numbering schemes, and/or data structures proposed particularly for tree structure of XML documents.

Naturally, each of the approaches has both keen advocates and objectors. The situation is not good especially for file system-based and object-oriented methods. The former ones suffer from inability of querying without an additional preprocessing of the data, whereas the latter approach fails especially in finding a corresponding efficient and comprehensive tool. Undoubtedly, the highest-performance techniques are the native ones, since they are proposed particularly for XML processing and do not need to artificially adapt existing structures to a new purpose. But the most practically used ones are methods which exploit features of "classical" (object-)relational databases. The reason is that (O)RDBMS are still regarded as universal data processing tools and their long theoretical and practical history can guarantee a reasonable level of reliability. 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. Thus until the

native XML methods "grow up", i.e. until there exists a reliable and robust implementation verified by years of both theoretic and implementation effort, it is still necessary to improve XML data management in (O)RDBMS.

Currently there is a plenty of existing works concerning database-based[1] XML data management. Almost all the major database vendors (such as, e.g., [14] [11] [12]) more or less support XML processing and even the SQL standard has been extended by a new part (SQL/XML) which introduces new XML data type and operations for XML data manipulation. The main concern of the database-based XML-processing techniques is the choice of the way XML data are stored into relations, so-called *XML-to-relational mapping*. Under a closer investigation the methods can be further classified and analyzed [50]. We usually distinguish *generic* (or *schema-oblivious*) methods which store XML data regardless the existence of corresponding XML schema (e.g. [35] [73] [43] [66] [32]), *schema-driven* methods based on structural information from existing schema of XML documents (e.g. [67] [64] [44] [49]), and *user-defined* methods which leave all the storage decisions in hands of users (e.g. [15] [9]).

Techniques of the first type usually view an XML document as a directed labeled tree with several types of nodes. (Hence, in fact, they also use a kind of document schema, though a quite general one.) We can further distinguish generic techniques which store purely components of the tree and their mutual relationship (e.g. [35]) and techniques which store additional structural information, usually using a kind of a numbering schema (e.g. [43]). Such schema enables to speed up certain types of queries, but usually at the cost of inefficient data updates. The fact that the techniques do not exploit possibly existing 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 information. But, together with the finding that a significant portion of real XML documents (52% [47] of randomly crawled or 7.4% [51] of semi-automatically collected[2]) have no schema at all, they seem to be the most practical choice.

By contrast, schema-driven methods have contradictory (dis)advantages. The situation is even worse for methods which are based particularly on XML Schema [70] [27] definitions (XSDs) and focus on their special features [49].

[1] In the rest of the text the term "database" represents an (O)RDBMS, if not explicitly stated alternatively.

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

As it is expectable, XSDs are used even less (only for 0.09% [47] of randomly crawled or 38% [51] of semi-automatically collected XML documents) and even if they are used, they often (in 85% of cases [26]) define so-called *local tree grammars* [54], i.e. languages that can be defined using DTD [28] as well. The most exploited "non-DTD" features are usually simple types [26] whose lack in DTD is 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 [51] shows that XML schemes are often too general, i.e. the set of document instances valid against a schema is much larger than the set of real-world ones. Excessive examples can be recursion or "*" operator which allow theoretically infinitely deep or wide XML documents. Naturally, XML schemes also cannot provide any information about, e.g., retrieval frequency of an element/attribute or the way they are retrieved. Thus not only XML schemes but also corresponding XML documents and XML queries need to be taken into account to get the 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. It is commonly offered by most known (O)RDBMS [21] as a feature that enables users to define what suits them most instead of being restricted by disadvantages of a particular technique. Nevertheless, the key problem is evident – it assumes that the user is skilled in both database and XML technologies and is able to specify an optimal database schema for a particular application.

Apparently, advantages of all three approaches are closely related to the particular situation. Thus it seem to be advisable to propose a method which is able to exploit the current situation or, at least, to comfort to it. If we analyze the database-based methods more deeply, we can distinguish so-called *flexible* or *adaptive* methods (e.g. [39] [62] [71] [75]). They take into account a given sample set of XML data and/or XML queries which specify the future usage and adapt the resulting database schema to them. Such techniques have naturally better performance results than the *fixed* ones, i.e. methods which use pre-defined set of mapping rules and heuristics regardless the intended future usage. A different set of methods, so-called *user-driven* ones (e.g. [34] [23]), also adapt the target schema to a particular

application. In this case it is determined by schema annotations which specify user-required local changes of a default fixed mapping strategy.

In this thesis we focus on further enhancing of adaptivity in the area of processing XML data using (O)RDBMS. Firstly, we perform a detailed analysis of existing works and especially remaining open issues. Secondly, we propose their enhancing which can be characterized as a hybrid user-driven adaptive mapping strategy, i.e. a combination of so-called user-driven and adaptive methods. In the enhancing we focus especially on deeper exploitation of user-given information, i.e. schema annotations, and we propose an approach which enables to identify new annotation candidates and thus to help users to find a more appropriate mapping strategy. For this purpose we propose a similarity measure which focuses mainly on structural similarity of the given data and an algorithm which enables reasonable tuning of its parameters on the basis of results of statistical analysis of real-world XML data. Using various experiments we show the behavior and efficiency of both the similarity measure and the hybrid mapping strategy on real XML data. And finally, we analyze the correctness and structure of the resulting mapping strategy and related query evaluation. We focus especially on the problem of correction of the set of annotation candidates, evaluation of parts of a single XML query using various storage strategies, and exploitation of redundancy. We conclude with a discussion of further possible improvements of the approach, as well as XML processing using (object-)relational databases in general.

1.1 Road Map

The rest of the thesis is structured as follows: In Chapter 2 we briefly state several basic definitions used in the rest of the text. Chapter 3 contains an analysis of existing adaptive mapping strategies and a discussion of corresponding open issues. In Chapter 4 we propose a possible enhancing of these methods – a hybrid user-driven adaptive method. We describe the approach mostly theoretically leaving implementation decisions and corresponding issues to the following chapters. In Chapter 5 we describe a similarity measure designed particularly for the purpose of the proposed hybrid method and especially the problem of reasonable tuning of its parameters. In the sixth chapter we provide an overview of statistical analysis of real-world XML data whose results were used several times in the work, in particular in the tuning

process and in experimental tests. Chapter 7 deals with the remaining issues concerning the correctness and structure of the resulting mapping strategy and related query evaluation. And finally, Chapter 8 provides conclusions and outlines possible future work and further improvements.

For better orientation in the text, each chapter is introduced with an abstract that briefly describes its content and lists papers where it was published, if there exists any.

Chapter 2

Definitions and Formalism

In this chapter we state basic terms used in the rest of the text. We use or slightly modify common notation and definitions from [29] and [26]. Other definitions, established particularly for the purpose of this thesis, are stated and explained in respective chapters.

An *XML document* is usually viewed as a directed labeled tree with several types of nodes whose edges represent relationships among them. Auxiliary 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,*
- $lab : V \rightarrow \Sigma_E \cup \Sigma_A \cup \Gamma$ *is a surjective function which assigns a label to each* $v \in V$ *s.t. for* $\forall\ e = \langle v_x, v_y \rangle \in E : (lab(v_x) \in \Sigma_E \wedge lab(v_y) \in \Sigma_E \cup \Sigma_A \cup \Gamma) \vee (lab(v_x) \in \Sigma_A \wedge lab(v_y) \in \Gamma)$, *and*
- r *is the root node of the tree.*

A node $v \in V$ is called an element *if* $lab(v) \in \Sigma_E$, an attribute *if* $lab(v) \in \Sigma_A$, *or* a text value *if* $lab(v) \in \Gamma$.

A *schema* of an XML document is usually defined using DTDs or XSDs which describe the allowed structure of an element using a regular expression called *content model*. An XML document is *valid* against a schema (or it is an *instance* of a schema) if each element matches its content model and has corresponding attributes. (We state the definitions for DTDs only for the text length. For XSDs are often used the same or similar ones.)

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* an empty content model, *pcdata denotes* a text content model, $f \in \Sigma'_E$, *","* *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.*

A content model α *s.t.* $\alpha \neq \epsilon \wedge \alpha \neq pcdata$ *is called* an element content model.

Definition 3 An XML schema S *is a four-tuple* (Σ'_E, Σ'_A, Δ, 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 a schema $S = (\Sigma'_E, \Sigma'_A, \Delta, s)$ *if* $lab(r) = s$ *and for* $\forall\ v \in V$ *and its label* e, *the sequence* $e_1\ e_2\ ...\ e_k$ *of labels of its subelements matches the regular expression* α, *where* $(e \rightarrow \alpha) \in \Delta$, *and the set of its attributes matches the set* $\beta \subseteq \Sigma'_A$, *where* $(e \rightarrow \beta) \in \Delta$.

To simplify the XML-to-relational mapping process an XML schema is often transformed into a graph representation. Probably its first occurrence, so-called *DTD graph*, can be found in [67]. There are also various other types of graph representation of an XML schema. 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* $G_S = (V, E, lab', r')$*, 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* $\forall\ v \in V$ *s.t.*
  - *If* $lab'(v_x) \in \Sigma'_E$:
    - $\{v_y | \langle v_x, v_y \rangle \in E \wedge lab'(v_y) \in \Sigma'_A\} = \beta$*, where* $(lab'(v_x) \rightarrow \beta) \in \Delta$ *and*
    - *the sequence of labels* $l_1\ l_2\ \ldots\ l_k$ *of nodes of content subgraph rooted at* v_x *in inorder traversal order forms the content model* α*, where* $(lab'(v_x) \rightarrow \alpha) \in \Delta$.
  - *If* $lab'(v_x) \in \Sigma'_A$ *then* $lab'(v_y) = pcdata$*, where* $e = \langle v_x, v_y \rangle \in E$.

  and
- r' *is the root node of the graph and* $lab'(r') = s$.

A content subgraph *of a schema graph* $G_S = (V, E, lab', r')$ *is its subgraph* $G'_S = (V', E')$ *rooted at node* $v_r \in V'$*, where* $lab'(v_r) \in \Sigma'_E$*, s.t.* $\forall$ *leaf node* $v \in V'$ *:* $lab'(v) \in \Sigma'_E \cup \{pcdata\}$ *and* $\forall$ *inner node* $v \in V'$ *:* $lab'(v) \in \{$ "|", "*", "+", "?", ","$\}$.

The core idea of XML-to-relational mapping methods is to *decompose* a given schema graph into *fragments*. Each fragment is then mapped to a corresponding relation.

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

Definition 7 A decomposition *of a schema graph* $G_S = (V, E, lab', r')$ *is a set of its fragments* $F = \{f_1, ..., f_n\}$*, where* $\forall\ v \in V$ *is a member of at least one fragment* $f_i \in F$*, where* $i = 1, 2, ..., n$.

For determining the best XML-to-relational strategy, the structure of the stored XML data, in particular the complexity of used content models, is usually analyzed. Basic characteristics simply distinguish the empty, text, and element content of an element, more complex ones involve their *depth* and width which is usually characterized by various types of *fan-out*. An inverse characteristic to fan-out is called *fan-in*.

Definition 8 A depth of a content model α *is inductively defined as follows:*
depth(ϵ) = 0;
depth(pcdata) = depth(f) = 1;
depth($\alpha_1, \alpha_2, ..., \alpha_n$) = *depth*($\alpha_1|\alpha_2|...|\alpha_n$) = *max*(*depth*($\alpha_i$)) + 1, *where* $i = 1, 2, ..., n$;
depth($\beta*$) = *depth*($\beta+$) = *depth*($\beta?$) = *depth*(β) + 1.

Definition 9 An element fan-out *of an element e is the number of distinct elements in content model* α, *where* $e \rightarrow \alpha$.
An attribute fan-out *of an element e is the cardinality of set* β, *where* $e \rightarrow \beta$.

Definition 10 A fan-in *of an element e is the cardinality of the set* $\gamma = \{f \mid f \rightarrow \alpha'$ *and the element e occurs in* $\alpha'\}$.
An element with large fan-in value is called a hub.

Note that the definitions of both depth and fan-out are specified for XML schemes but can be used (in slightly modified versions) also for XML documents. On the other hand, the fan-in value is in case of XML documents senseless since it is for all non-root elements equal to 1. In addition to depth, in case of XML documents we can also speak about *levels*. Note that this construct can be (and often is) specified also for XML schemes.

Definition 11 A level of an element *e is the distance of its node from the root node r. The level of the root node is 0.*
A distance of elements e_x *and* e_y *is the number of edges in document tree T separating their corresponding nodes.*

Last but not least, there are characteristics which describe special types of content models. Probably the most common ones are *recursive* and *mixed* content. In addition, in case of XSDs we can distinguish also the *unordered* content specified by the `all` operator.

Definition 12 *An element e is* recursive *if e is reachable from e.*

A node e' is reachable *from e if there exists a directed path from e to e' in schema graph G_S.*

The element-descendant association is called an ed-pair*.*

Definition 13 *A content model α is* mixed*, if $\alpha = (\alpha_1|...|\alpha_n|pcdata)* \mid (\alpha_1|...|\alpha_n|pcdata)+$ where $n \geqslant 1$ and for $i = 1, 2, ..., n$ content model $\alpha_i \neq \epsilon \wedge \alpha_i \neq pcdata$.*

An element e is called mixed-content element *if content model α, where $e \rightarrow \alpha$, is mixed.*

Chapter 3

Related Work

In this chapter we study techniques which are currently considered as the most efficient way of XML processing based on (object-)relational databases, so-called adaptive or flexible mapping methods. 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 open issues.

Most of the contents of this chapter has been published in the following papers:

Mlýnková, I. – Pokorný, J.: Adaptability of Methods for Processing XML Data using Relational Databases – the State of the Art and Open Problems. RCIS '07: Proceedings of the 1st International Conference on Research Challenges in Information Science, pages 183 – 194, Ouarzazate, Morocco, April 2007. Ecole Marocaine des Sciences de l'Ingénieur, 2007. (Note: The Best Paper Award; selected for publishing in Special Issue of the International Journal of Computer Science and Applications, ISSN 0972-9038, Volume 4, Issue 2, pages 43 – 62, Technomathematics Research Foundation, July 2007.)

Mlýnková, I. – Pokorný, J.: XML in the World of (Object-)Relational Database Systems. ISD '04: Proceedings of the 13th International Conference on Information Systems Development, pages 63 – 76, Vilnius, Lithuania, September 2004. Springer Science+Business Media Inc., 2005. ISBN 978-0-387-25026-7.

Up to now several papers have focused on a proposal of an adaptive database-based XML-processing method. We distinguish two main directions – *cost-driven* and *user-driven*. Techniques of the former group can choose the

most efficient XML-to-relational storage strategy automatically. They usually evaluate a set of possible mappings and choose the optimal 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 (see Introduction), since their approach is different: By default they apply a fixed mapping, but users can influence the mapping process by annotating fragments of the input XML schema with demanded storage strategies. Similarly to the user-defined techniques this approach also assumes a skilled user, but most of the work is done by the system itself. The user is expected to help the mapping process, not to perform it.

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. Thus 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 the following five features:

1. an initial XML schema S_{init},
2. a set of XML schema transformations $T = \{t_1, t_2, ..., t_n\}$, where for $i = 1, 2, ..., n : t_i$ transforms a given schema S into a schema S_i,
3. a fixed XML-to-relational mapping function f_{map} which transforms a given XML schema S into a relational schema R,
4. a set of sample data D_{sample} characterizing the future application, which usually consists of a set of XML documents $\{d_1, d_2, .., d_k\}$ valid against S_{init}, and a set of XML queries $\{q_1, q_2, .., q_l\}$ over S_{init}, eventually with corresponding weights $\{w_1, w_2, .., w_l\}$, where for $i = 1, 2, ..., l : w_i \in [0, 1]$, and
5. a cost function f_{cost} which evaluates the cost 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.

A naive but illustrative cost-driven storage strategy that is based on the idea of using a "brute force" is depicted by Algorithm 1. It 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 $R_{opt} = f_{map}(s)$.

Algorithm 1 Naive Search Algorithm

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

```
1:  S ← {S_init}
2:  while ∃ t ∈ T, s ∈ S s.t. t(s) ∉ S do
3:      S ← S ∪ {t(s)}
4:  end while
5:  cost_opt ← ∞
6:  for all s ∈ S do
7:      R_tmp ← f_map(s)
8:      cost_tmp ← f_cost(R_tmp, D_sample)
9:      if cost_tmp < cost_opt then
10:         R_opt ← R_tmp
11:         cost_opt ← cost_tmp
12:     end if
13: end for
14: return R_opt
```

Obviously the complexity of the algorithm depends strongly 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 [71] [75] [41]. Thus the existing techniques in fact search for a suboptimal solution using various heuristics, greedy strategies, approximation algorithms, terminal conditions, etc. We can also observe that 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 $\forall\ R$.

3.1.1 Hybrid Object-Relational Mapping

One of the first attempts of a cost-driven adaptive approach is a method called *Hybrid object-relational mapping* [39]. It is based on the fact that if XML documents are mostly semi-structured, a "classical" decomposition of less structured XML parts into relations leads to inefficient query processing caused by plenty of join operations. The algorithm exploits the idea of storing well structured parts into relations and semi-structured parts 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 [67], whereas, in addition, it exploits NF^2-relations using constructs such as `set-of`, `tuple-of`, and `list-of`. Therefore the main concern of the method is to identify the structured and semi-structured parts. It consists of the following steps:

1. A schema graph $G_1 = (V_1, E_1, lab'_1, r'_1)$ is built for a given DTD[1].
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, where $i = 1, 2, ..., k : \omega_v < \omega_{LOD}$ 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.
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 strategy.

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

$$\omega_v = \frac{1}{2}\omega_{S_v} + \frac{1}{4}\omega_{D_v} + \frac{1}{4}\omega_{Q_v} = \frac{1}{2}\omega_{S_v} + \frac{1}{4} \cdot \frac{card(D_v)}{card(D)} + \frac{1}{4} \cdot \frac{card(Q_v)}{card(Q)} \quad (3.1)$$

[1] Assuming that the DTD is a directed tree.

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 (see [39]), $D \subseteq D_{sample}$ 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.

As we can see, the algorithm optimizes the naive approach mainly by the facts that the schema graph is preprocessed, i.e. ω_v is determined for $\forall\, v \in V_1$, that the set of transformations T is a singleton, and that the transformation is performed if the current node satisfies the above mentioned conditions (a) – (c). The preprocessing ensures that the complexity of the search algorithm is given by $K_1 \cdot card(V_1) + K_2 \cdot card(E_1)$, where $K_1, K_2 \in \mathbb{N}$. But, on the other hand, the optimization is too restrictive in terms of the amount of possible XML-to-relational mappings.

3.1.2 FlexMap Mapping

Another example of adaptive cost-driven methods was implemented as so-called *FlexMap framework* [62]. The algorithm optimizes the naive approach using a simple greedy strategy as depicted in Algorithm 2. 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 the following operations:

- *Inlining and outlining* – inverse operations which enable to store columns of a subelement/attribute either in parent table or in a separate table
- *Splitting and merging elements* – inverse operations which enable to store a shared element[2] either in a common table or in separate tables
- *Associativity and commutativity*
- *Union distribution and factorization* – inverse operations which enable to separate out components of a union using equation $(a, (b|c)) = ((a, b)|(a, c))$

[2] An element with multiple parent elements in the schema – see [67].

Algorithm 2 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
```

- *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, whereas simplifying unions generates a more general schema, i.e. a schema with larger set of valid document instances. (However, only inlining and outlining were implemented and experimentally tested by the FlexMap system.)

The fixed mapping again uses a strategy similar to the Hybrid algorithm [67] but it is applied locally on each fragment of the schema 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 construction of a particular relational schema, loading sample XML data into the relations, and cost analysis of the resulting relational structures. The FlexMap evaluation exploits an XML Schema-aware statistics framework *StatiX* [36] which analyzes the structure of a given XSD and XML documents and computes their statistical summary, which is then "mapped" to relational statistics regarding the fixed XML-to-relational mapping. Together with sample query workload they are used as an input for a classical relational optimizer which estimates the resulting cost. Thus no relational schema has to be constructed and, as the statistics are respectively updated at each XML-to-XML transformation, the XML documents need to be processed only once.

3.1.3 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 [71] as an *Adjustable and adaptable method (AAM)*. In this case the authors adapt the given problem to features of genetic algorithms. It 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 schema fragments. 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 [35]. 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 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 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} is expressed as:

$$\begin{aligned} f_{cost}(R, D_{sample}) &= f_M(R, D_{sample}) + f_Q(R, D_{sample}) = \\ &= \sum_{l=1}^{q} C_l * R_l + (\sum_{i=1}^{m} S_i * P_{S_i} + \sum_{k=1}^{n} J_k * P_{J_k}) \end{aligned} \tag{3.2}$$

where

- f_M is a *space-cost function*, where C_l is the number of columns and R_l is the number of rows in table T_l created for l-th element in the schema and q is the number of all elements in the schema, and

- f_Q is a *query-cost function*, where S_i is cost and P_{S_i} is probability of i-th select query and J_k is cost and P_{J_k} is probability of k-th 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 represents the total memory cost of the mapping instance, whereas f_Q 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. Also note that 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.

3.1.4 Hill Climbing Algorithm

The last but not least cost-driven adaptive representative can be found in paper [75]. The approach is again based on a greedy type of algorithm, in this case a *Hill climbing strategy* that is depicted by Algorithm 3.

Algorithm 3 Hill Climbing Algorithm

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

1: $S_{opt} \leftarrow S_{init}$
2: $R_{opt} \leftarrow f_{map}(S_{opt})$
3: $cost_{opt} \leftarrow f_{cost}(R_{opt}, D_{sample})$
4: $T_{tmp} \leftarrow T$
5: **while** $T_{tmp} \neq \emptyset$ **do**
6: $t \leftarrow$ any member of T_{tmp}
7: $T_{tmp} \leftarrow T_{tmp} \backslash \{t\}$
8: $S_{tmp} \leftarrow t(S_{opt})$
9: $cost_{tmp} \leftarrow f_{cost}(f_{map}(S_{tmp}), D_{sample})$
10: **if** $cost_{tmp} < cost_{opt}$ **then**
11: $S_{opt} \leftarrow S_{tmp}$
12: $R_{opt} \leftarrow f_{map}(S_{tmp})$
13: $cost_{opt} \leftarrow cost_{tmp}$
14: $T_{tmp} \leftarrow T$
15: **end if**
16: **end while**
17: **return** R_{opt}

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, $\langle u, v \rangle$) – cuts fragment f into fragments f_1 and f_2, s.t. $f_1 \cup f_2 = f$, where $\langle u, v \rangle$ 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, $\langle u, v \rangle$) – splits fragment f into twin fragments f_1 and f_2 horizontally from edge $\langle u, v \rangle$, where $u \notin f$ and $v \in f$, s.t. $ext(f_1) \cup ext(f_2) = ext(f)$ and $ext(f_1) \cap ext(f_2) = \emptyset$ [3] [4]
- *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. Conversely, *H-Cut* operation corresponds to splitting of elements used in FlexMap mapping, i.e. duplication of the shared part, and 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 the root element of f_i. Note that such mapping is again similar to locally applied Universal table [35].

The cost function f_{cost} is expressed as:

$$f_{cost}(R, D_{sample}) = \sum_{i=1}^{n} w_i \cdot cost(q_i, R) \tag{3.3}$$

where D_{sample} consists of a sample set of XML documents and a given query workload. The cost function $cost(q_i, R)$ for a query q_i which accesses fragment set $\{f_{i1}, ..., f_{im}\}$ is expressed as:

$$cost(q_i, R) = \begin{cases} |f_{i1}| & m = 1 \\ \sum_{j,k} (|f_{ij}| \cdot Sel_{ij} + \delta \cdot (|E_{ij}| + |E_{ik}|)/2) & m > 1 \end{cases} \tag{3.4}$$

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}.

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

[4] 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.

The authors further analyze the influence of the choice of initial schema S_{init} on efficiency of the search algorithm. They use three types of initial schema decompositions leading to Binary [35], Shared [67], or Hybrid 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 before, the most flexible approach is the *user-defined* mapping, i.e. the idea "to leave the whole process in hands of a user" who defines both the target database schema and the required mapping. Due to simple implementation it is supported in most commercial database systems [21]. At first sight the idea is correct – users can decide what suits them most and are not restricted by disadvantages of a particular technique. The problem is that such approach assumes users skilled in two complex technologies and for more complex applications the design of an optimal relational schema is an uneasy task in general.

On this account new techniques – in this thesis 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 annotations is naturally limited but still enough powerful to define various mapping strategies.

Each of the techniques is characterized by the following four features:

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

3.2.1 ShreX Framework

Probably the first approach which faces the mentioned issues is proposed in paper [34] as a mapping definition framework called *ShreX*. It allows users

to specify the required mapping, checks its *correctness* and *completeness* and completes possible incompleteness using default rules. The mapping specifications are made by annotating the input XSD with a predefined set of attributes Ω_A listed in Table 3.1.

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.
`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 [35].
`maptoclob`	attribute or element	`true`, `false`	If the value is `true`, the element/attribute is mapped to a CLOB column.

Table 3.1: Annotation attributes for ShreX

The set of allowed XML-to-relational mappings $\{f^i_{map}\}_{i=1,2,\ldots,n}$ involves inlining and outlining of an element/attribute, Edge mapping strategy, and mapping an element or an 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 integer 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.

3.2.2 XCacheDB System

Paper [23] also proposes a user-driven mapping strategy which is implemented and experimentally tested as an *XCacheDB system* that considers only unordered and acyclic XML schemes and omits mixed-content elements. The set of annotating attributes Ω_A that can be assigned to any node $v \in S_{init}$ is listed in Table 3.2.

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 3.2: Annotation attributes for XCacheDB

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 is an atomic node[5], 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 ID type which points to ID of v.

In general, the set of possible mapping strategies supported by the system can be characterized as a set of modifications of a single mapping strategy.

3.3 Theoretic Issues

Besides proposals of cost-driven and user-driven techniques, there are also papers which discuss the corresponding open issues on theoretic level.

3.3.1 Data Redundancy

As mentioned above, the XCacheDB system allows a certain degree of redundancy, in particular duplication into BLOB columns and the violation of BCNF or 3NF condition. The paper [23] discusses the strategy also on theoretic level and defines four classes of XML schema decompositions. Before we state the definitions we have to note that the approach is based on a slightly different graph representation than in Definition 5. In particular, the nodes of the graph correspond to elements, attributes, or pcdata, whereas edges are labeled with corresponding operators.

Definition 14 *A schema* decomposition is minimal *if all edges connecting nodes of different fragments are labeled with "*" or "+".*

Definition 15 *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 16 *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.*

[5] An attribute node or an element node having text content.

Definition 17 *A schema* decomposition is inlined *if it is non-MVD but it is not a 4NF decomposition. A* fragment is inlined *if it is non-MVD but it is not a 4NF fragment.*

According to these definitions, fixed mapping strategies (such as, e.g., [67] [49]) 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 [23] 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.

3.3.2 Grouping Problem

Paper [41] 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*[6] 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 (so-called *fully partitioned strategy*). For analysis of its complexity the authors define two simple cost metrics:

- *RelCount* – the cost of a relational query is the number of relation instances in the query expression
- *RelSize* – the cost of a relational query is the sum of the number of tuples in relation instances in the query 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*[7] is converted into union of several queries, one for each satisfying wild-card substitution

[6] An algorithm for translating XML queries into SQL queries

[7] A query containing "`//`" or "`/*`" operators.

- *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.

3.4 Summary

We can sum up the state of the art of adaptability of database-based XML-processing methods into the 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 often 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. The 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 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 the relational schema, loading 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 exploit the user-given hints as much as possible.

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.

3.5 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 adaptive methods. We describe and discuss them in this chapter starting from the least complex ones.

Missing Input Data As we already know, for cost-driven techniques there are three types of input data – an XML schema S_{init}, a set of XML documents $\{d_1, d_2, .., d_k\}$, and a set of XML queries $\{q_1, q_2, .., q_l\}$. The problem of missing schema S_{init} was already outlined in the Introduction in connection with (dis)advantages of generic and schema-driven methods. As we suppose that the adaptability is the ability to adapt to the given situation, a method which does not depend on existence of an XML schema but can exploit the

information if being given is probably a natural first improvement. This idea is also strongly related to the mentioned problem of choice of a locally good initial schema S_{init}. The corresponding questions are:

- Can be the user-given schema considered as a good candidate for S_{init}?
- How can we 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. [53] [55]). Assuming that documents are more precise sources of structural information, we can expect that a schema generated on their bases will have better characteristics too.

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. [47] [51]). Furthermore, the surveys show that real XML data are surprisingly simple in comparison with the expressive power of languages for schema definition and thus the default mapping strategy does not have to be complex too. It should rather focus on efficient processing of frequently used XML patterns.

Finally, 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 collecting such real representatives is not as straightforward as in case of XML documents. Currently the best sources of XML queries are XML benchmarking projects (e.g. [61] [72]) 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 of particular application. Naturally, the query statistics can be gathered by the system itself and the schema can be adapted continuously, as discussed later in the text.

Efficient Solution of Subproblems A surprising fact we have encountered are numerous simplifications of the chosen solutions. As 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. [69]) and that adaptive methods should cope with various situations. In addition,

both the often omitted constructs are in real-world XML schemes used more often than it is usually assumed [51].

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 strategy for not annotated parts of the schema is again a fixed one. It can be an interesting optimization to join the ideas and search the (sub)optimal mapping for not annotated parts using a cost-driven method.

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

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

A possible answer can be found in the idea of pattern matching, i.e. to use the user-given schema annotations as "hints" how to store particular XML patterns. We can naturally assume 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 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 equivalence of XML schema fragments and algorithms how to determine them. Last but not least, such system should focus on scalability of the similarity metric and particularly its reasonable default setting.

Theoretical Analysis of the Problem As the overview shows, there are various types of XML-to-XML transformations, whereas 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 is the question of NP-hardness in connection with the set of allowed transformations and its complexity (similarly to paper [41] which analyzes theoretical complexity of

combinations of cost metrics and query translation algorithms). Such survey would provide useful information especially for optimizations of the search algorithm.

Dynamic Adaptability The last but not least issue is connected with the most striking disadvantage of adaptive methods – the problem of possible changes of XML queries or XML data that can lead to crucial worsening of the efficiency. As mentioned above, it is also related to the problem of missing input XML queries and ways how to gather them. The question of changes of XML data opens also 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. On the other hand, this "brute-force" approach can serve as an inspiration. Supposing that changes especially in case of XML queries will not be radical, the modifications 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 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.

Chapter 4

Hybrid User-Driven Adaptive Method

In this chapter we introduce a method which can be characterized as a hybrid user-driven adaptive mapping strategy focusing on two persisting disadvantages of user-driven methods. Firstly, it is the fact that the default mapping strategy is (to our knowledge) always a fixed one. Since the corresponding system must be able to store schema fragments in various ways, an adaptive enhancing of the fixed method seems to be quite natural and suitable. The second shortcoming is weak exploitation of the user-given information. The annotations a user provides can not only be directly applied on particular schema fragments, but can be regarded as "hints" how to store particular XML patterns. 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.

Most of the contents of this chapter has been published in the following papers:

Mlýnková, I.: A Journey towards More Efficient Processing of XML Data in (O)RDBMS. To appear in CIT '07: Proceedings of the 7th IEEE International Conference on Computer and Information Technology, Fukushima, Japan, October 2007. IEEE Computer Society, 2007.

Mlýnková, I.: An XML-to-Relational User-Driven Mapping Strategy Based on Similarity and Adaptivity. SYRCoDIS '07: Proceedings of the 4th Spring Young Researchers Colloquium on Databases and Information Systems, pages 9 – 20, Moscow, Russian Federation, May 2007. CEUR Workshop Proceed-

The key concern of our approach 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 [18] fragments. A classical fixed schema-driven mapping strategy 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 [51]), 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 – the updatability of 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 various 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 the 4NF schema decomposition into relations [23] which can be achieved, e.g., using the Hybrid algorithm [67], 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 (see Chapter 3.3.1).

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 a default mapping. But although there is a plenty of existing representatives of the two approaches (as we have described in Chapter 3), there are still numerous weak points and open issues that should be improved and solved.

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 the proposal is how to identify identical or similar fragments within the schema.

The 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 the 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.

4.1 Proposed Algorithm

A general idea of fixed schema-driven XML-to-relational mapping methods is to *decompose* the given XML schema $S = (\Sigma'_E, \Sigma'_A, \Delta, s)$ into a set of

relations $R = \{r_1, r_2, ..., r_n\}$ using a mapping strategy s_{rel}. An extreme case is when S is decomposed into a single relation resulting in many null values. Other extreme occurs when for each element $e \in \Sigma'_E$ a single relation is created resulting in numerous join operations. (Note that since fixed mapping methods view an XML document as general directed tree with several types of nodes, we can speak about schema decomposition too.)

In user-driven strategies the decomposition is influenced by user-defined *annotations* which specify how a particular user wants to store selected schema fragments $F = \{f_1, f_2, ..., f_m\}$. The user usually provides S (i.e. selected elements determining the fragments) with *annotating attributes* from the predefined set of attribute names Ω_A, each of which represents a particular fixed mapping strategy, resulting in an *annotated schema* S'. A classical user-driven strategy then consist of the following steps:

1. S is annotated using Ω_A resulting in S'.
2. Annotated fragments from F are decomposed according to appropriate mapping methods.
3. Not annotated fragments of S' are decomposed using a default fixed mapping strategy s_{def}.

The proposed approach enhances a classical user-driven strategy combining it with the idea of adaptivity. We simply add the following steps between the step 1 and 2:

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

b. For $\forall\ f \in F$ all fragments in F_f are annotated with annotating attributes of f.

c. $S \backslash F$ is annotated using an adaptive strategy.

The whole mapping process is schematically depicted in Figure 4.1 where the given schema S with $F = \{f, g\}$ is mapped to a database schema R. If the proposed enhancing, i.e. steps 1.a – 1.c, are 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

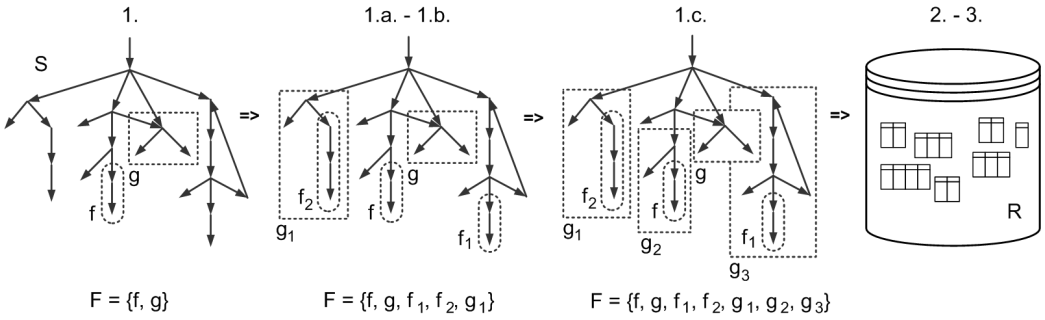


Figure 4.1: Schema of the mapping process

and g are annotated using user-required strategies and the rest of the schema using s_{def}.

As it is obvious, the basic ideas are relatively simple. But if we analyze the strategies more deeply, several interesting issues and open problems that need to be solved occur. We deal with them in the following chapters.

4.1.1 Searching for Similar Fragments

Considering the idea of searching for similar fragments in more depth there are several open issues, in particular the definition of annotated fragments, the supported types of annotations (i.e. the fixed mapping strategies), the measure of similarity, and the search algorithm. In addition, all these aspects mutually influence each other.

Annotated Fragments

Firstly, for easier processing we view an XML schema S, no matter if annotated or not, as a graph (see Definition 5). Next, we assume that each annotated fragment $f_e \in F$ is uniquely determined by an element e annotated using an annotating attribute $a \in \Omega_A$.

Definition 18 *An* element fragment f_e *of a schema* S *is each subgraph of* G_S *consisting of an element* e*, all nodes reachable from* e*, and corresponding edges.*

Φ *is a set of all element fragments of* S*.*

Definition 19 *An* annotated fragment f_e *of schema* S *is an element fragment of* S *rooted at an annotated element* e *excluding all annotating attributes from* Ω_A.

An annotated element e *of schema* S *is an element provided with an annotated attribute from* Ω_A.

Note the difference between a general schema fragment (see Definition 6) and an element fragment (see Definition 18). In the rest of the text we use the term fragment for simplicity, referring to an element fragment if not explicitly stated alternatively. Also note that this definition allows to annotate only element definitions. It can be reasonable to annotate also single attributes, attribute groups, or groups of elements (considering XSDs) similarly to [34], but we restrict to elements for easier description.

As we want to support shared elements and recursion, since both the constructs are widely used in real XML data [51], 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 20 *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 4.2. A shared element is highlighted using a dotted rectangle, a recursive element is highlighted using a dotted circle.

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. In addition, the real implementation does not have to perform the duplication of the shared fragments in fact. 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 [51]. 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.

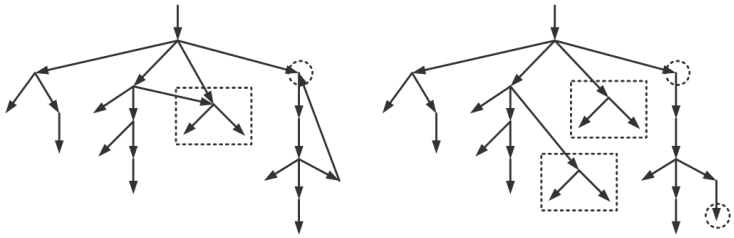


Figure 4.2: A schema graph G_S and an expanded schema graph G_S^{ex}

Types of Annotations

From Definitions 19 and 20 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 $f_y \subseteq f_x$.*

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 on the common schema fragment.

Definition 21 *Intersecting annotations are* redundant *if the corresponding mapping strategies are applied on the common schema fragment separately.*

Definition 22 *Intersecting annotations are* overriding *if only one of the corresponding mapping strategies is applied on the common schema fragment.*

Definition 23 *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 alternatively. 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.

Each subset of supported annotations is assigned a (user-specified) intersection type for particular orders of their compositions. This can involve plenty of specifications, but, in fact, the amount of reasonable and thus necessary specifications is much lower than the theoretically allowed ones. We deal with this topic in Chapter 7 in detail.

Note that the existing systems [34] [23] mostly support overriding and influencing annotations, the XCacheDB system [23], in addition, supports a kind of redundant intersection similar to the above described example.

Similarity Measure and Search Algorithm

The main idea of the enhancing remains the same regardless the chosen similarity measure and search algorithm. The choice of the measure influences the precision of the system, whereas the algorithm influences the efficiency of finding the required fragments. In general there are not many ways how to avoid the exhaustive search. And although we can assume that $card(F) = m$ is small, i.e. that a user annotates only several fragments, the exhaustive search can be expensive due to the size of G_S. Therefore, for the purpose of optimization, we exploit characteristics of the similarity measure.

Similarly to most of existing algorithms [45] [33] for measuring similarity on schema level we use various supplemental *matchers* [60], i.e. functions which evaluate similarity of a particular feature of the given schema fragments, such as, e.g., similarity of depths, similarity of number of elements, similarity of fan-outs, etc.

Definition 24 *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 25 *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 into the resulting composite similarity value. The most common and verified [33] way of composition is usually a kind of weighted sum.

Definition 26 *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.*

For most of the usually used matchers the knowledge of actual value of the analyzed feature for child nodes is necessary for evaluating the value for their parent node. Thus the search algorithm uses a bottom-up strategy, i.e. starting from leaf nodes towards the root node, and searches for schema fragments exceeding a given threshold $T_{sim} \in [0,1]$. The question is whether we should annotate all schema fragments exceeding T_{sim}. Let us consider the situation depicted in Figure 4.3, where for $i \in \{1,2,3\}$ $sim(f, f_i) > T_{sim}$ and f is the annotated fragment.

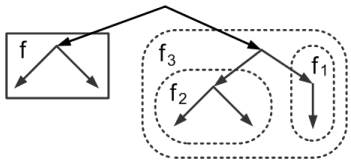


Figure 4.3: Similar fragments on the same root path

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*[1]. 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 4.3 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

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

reasonable and natural to annotate fragments f_1 and f_2 rather than whole f_3. If we generalize the idea, the algorithm annotates an element e using annotations of each fragment $f \in F$ s.t. $sim(f, f_e) > T_{sim}$ and $\nexists$ element e' on any root path traversing e s.t. $sim(f, f_{e'}) > sim(f, f_e)$.

Therefore, for the purpose of optimization we need to know the behavior of the similarity function on particular root paths. In the optimal case if we knew 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 similarity value. A sample situation can be seen in Figure 4.4 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 $f_5, f_6, ..., f_r$.

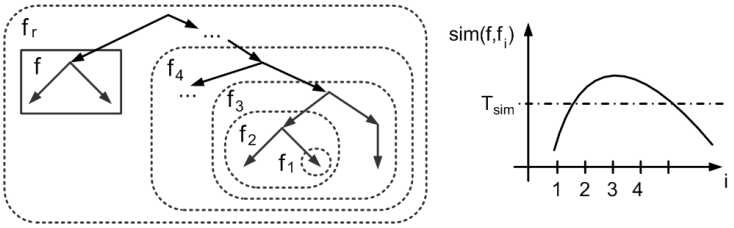


Figure 4.4: Exploitation of behavior of similarity function

Although we can hardly ensure that m_{comp} is concave, we can assume that at least q of the matchers, where $1 \leqslant q \leqslant p$, have this property. For instance a trivial matcher with such behavior can compare the number of distinct element or attribute names, the number of similar operators, the depth of the corresponding content model, etc. Such information is then used as heuristics 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.

The whole optimization of the approach, so-called *basic annotation strategy (BAS)*, is depicted by Algorithm 4, 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 $\langle fragment, similarity \rangle$, i.e. references to fragments (and corresponding similarity values) within its subtree that are candidates for annotation.

Algorithm 4 Basic Annotation Strategy (BAS)

Input: S, F, m_1, m_2, ..., m_q, m_{q+1}, ..., m_p, m_{comp}, T_{sim}
Output: $F \cup$ newly annotated fragments

```
1:  F' ← F
2:  for all f ∈ F do
3:    listToProcess ← leaf elements of G_S^ex \ {f}
4:    listOfProcessed ← ∅
5:    while listToProcess ≠ ∅ do
6:      for all e ∈ listToProcess do
7:        e.candidates ← ∅
8:        f_e ← subgraph rooted at e
9:        sim_e ← m_comp(f, f_e)
10:       for all c ∈ e.subelems do
11:         for all ⟨f', sim⟩ ∈ c.candidates do
12:           if sim > sim_e then
13:             e.candidates ← e.candidates ∪ {⟨f', sim⟩}
14:           end if
15:         end for
16:       end for
17:       if e.candidates = ∅ ∧ sim_e > T_sim then
18:         e.candidates ← e.candidates ∪ {⟨f_e, sim_e⟩}
19:       end if
20:       if terminate(f, e, m_1, m_2, ..., m_q, T_sim) then
21:         for all ⟨f', sim⟩ ∈ e.candidates do
22:           f'.annotation ← f.annotation
23:           F' ← F' ∪ {f'}
24:         end for
25:       else
26:         if ∀ s ∈ e.siblings : s ∈ listOfProcessed then
27:           listToProcess ← listToProcess ∪ {e.parent}
28:         end if
29:       end if
30:       listToProcess ← listToProcess \ {e}
31:       listOfProcessed ← listOfProcessed ∪ {e}
32:     end for
33:   end while
34: end for
35: return F'
```

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 heuristics 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.

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 heuristics and for the composite similarity measure. Nevertheless, we deal with the above described ones, since it is the typical and verified way for evaluating similarity among XML schemes. Also note that since it is natural that a user does not want to annotate all similar fragments in a similar way, we assume that such fragments are denoted as *final* and excluded from the processing.

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. We exploit the results of analysis of real-world XML data (see Chapter 6) which shows two important aspects:

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

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 do 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 viewed as misleading. We use the former option assuming that the tuning process (see Chapter 5) sets appropriate weights to all the matchers including the recursive one.

4.1.2 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 BAS algorithm. As the second enhancing we apply an adaptive mapping strategy on the remaining parts of the schema. At first glance the user-driven techniques have nothing in common with the adaptive ones. But 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 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 27 *A* contraction *of a schema graph G_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_S^{con}.*

The basic idea of the adaptive strategy is as follows: Having a contracted graph G_S^{con} we repeat the BAS algorithm and operation contraction until there is no fragment to annotate. The BAS algorithm is just slightly modified:

- It searches for schema fragments which are not involved in the schema, i.e. it searches among all nodes of the given graph and returns the (eventually empty) set of identified fragments.
- For similarity evaluation we do not take into account contracted nodes.

- The annotations of contracted nodes are always overriding in relation to the newly defined ones.

We denote this modification of BAS as a *contraction-aware annotation strategy (CAS)*. The resulting annotating strategy, so called *global annotation strategy (GAS)*, is depicted by Algorithm 5, where function *contract* applies operation contraction on graph of the given schema S and set of fragments F and function *restore* restores all the contracted nodes of the given schema to the original ones.

Algorithm 5 Global Annotation Strategy (GAS)

Input: S, F, m_1, m_2, ..., m_p, m_{comp}, T_{sim}
Output: $F \cup$ newly annotated fragments

```
1: F' ← BAS(S, F, m_1, m_2, ..., m_p, m_comp, T_sim)
2: F^tmp ← F'
3: while F^tmp ≠ ∅ do
4:    contract(S, F^tmp)
5:    F^tmp ← CAS(S, F, m_1, m_2, ..., m_p, m_comp, T_sim)
6:    F' ← F' ∪ F^tmp
7: end while
8: restore(S, F')
9: return F'
```

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.

4.2 Experimental Implementation

For testing the key features of the proposed approach we have implemented an experimental system called *UserMap*. In the following experiments we analyze the key aspect of the proposed approach – the BAS and GAS algorithms and their behavior on real data. We use the same 98 real-world XML schemes that we used in statistical analysis described in Chapter 6 divided into database (`dat`), document (`doc`), exchange (`ex`), report (`rep`), and research (`res`) category. Their basic characteristics can be seen in Tables 6.3 and 6.4. The first two categories are similar to classical data-centric

and document-centric ones, the other three are introduced in the analysis to enable finer division. In experiments we use a slight modification of GAS (Algorithm 5) which enables to compare its behavior within the categories. In particular, the annotated fragments are represented using a separate testing set of schema fragments consisting of 5 data-centric, 5 document-centric, 3 relational, and 3 DNA real-world schema fragments (see Definition 32 and 31). Table 4.1 shows results of characteristics of the algorithm applied on all the sample fragments per each category.

Characteristic	**dat**	**doc**	**ex**	**rep**	**res**
Average number of iterations	2.7	3.9	2.9	4.1	4.3
Average % of not annotated nodes	2.1	53.4	13.5	25.6	31.1
% of fully contracted schemes	93.7	22.2	81.1	0.0	28.6

Table 4.1: General characteristics per category

As we can see, the algorithm has quite reasonable behavior. Firstly, the number of iterations is not an extreme one – the algorithm is able to perform more than one contraction (i.e. not only the BAS algorithm is applied) and, on the other hand, there are no extreme values with regard to usual depth or number of elements in the schemes (see Table 6.3). From the other two characteristics it is obvious that the schemes are not usually fully contracted, i.e. the storage strategies are not determined for the whole schema (although it depends highly on the particular category). This indicates that the default mapping strategy s_{def} should be still specified. If we compare the average number of iterations with the percentage of fully contracted schemes, it is surprising that schemes with the lower amount of contractions are fully contracted more often. It is probably caused by the fact, that the two categories, i.e. `dat` and `ex`, usually contain schemes with much simpler structure than, e.g., the `doc` one, or much regular than, e.g., the `res` one.

Next set of performed tests analyzed the behavior of the algorithm in particular iterations, especially the percentage of annotated nodes at each iteration, as depicted by graphs in Figure 4.5.

As we can observe, the percentage of annotated nodes is usually highest in the first iteration, i.e. using the BAS algorithm, and then, with the decreasing number of nodes, rapidly decreases too. The only exceptions are the `rep` category, where the percentage grows up to third iteration and the `res` category, where it later slowly grows up to seventh iteration. It is probably

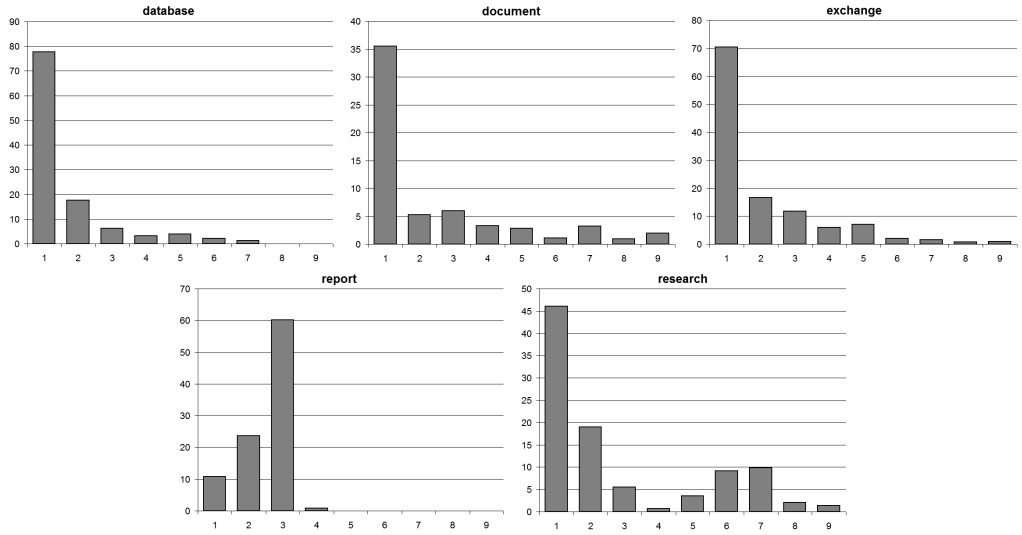


Figure 4.5: Average percentage of annotated nodes at each iteration

caused by less regular structure than in the other three cases, especially in case of `res` category, as well as the lower number of sample XML schemes and thus less precise tuning of the similarity measure (see Chapter 5.3.2).

In the last set of performed tests we have analyzed the relationship between types of schema fragments and the iterations. The finding is that various types of schema fragments appear "randomly" regardless the iteration. This indicates that the algorithm does not provide degenerated schemes, such as, e.g., a schema where all the annotations correspond to a single sample schema fragment.

4.3 Conclusion

The results of the experiments show that the proposed approach is able to exploit the user-given information more deeply and find more appropriate mapping strategy for not annotated schema parts than s_{def}. When applied on real-world XML schemes and schema fragments, the algorithm behaves quite reasonably, though it is not usually able to annotate the given schema fully. This indicates that the default mapping strategy s_{def} is still important.

Obviously there are several open issues related to the approach. Firstly, it is highly dependent on the choice and precision of the similarity measure.

We deal with this problem in Chapter 5 exploiting the results of statistical analysis described in Chapter 6. On the other hand, since the approach is proposed to enable a user to assign a mapping strategy to a chosen schema fragment which is suitable for the actual application, though it can be highly inefficient in the general case, an analysis of efficiency of the resulting storage strategy would be quite predictable and thus useless. But, on the other hand, there remains the open issue of the correctness and structure of the resulting mapping strategy and related query evaluation. We deal with these issues in Chapter 7.

Chapter 5

Similarity Measure

In this chapter we describe a similarity measure designed primarily for the purpose of enhancing of user-driven XML-to-relational storage strategies described in Chapter 4. In comparison with the existing approaches the method differs mainly in two aspects: Firstly, it focuses on structural similarity of the given schema fragments instead of semantics of element/attribute names, context of the analyzed schema fragments, etc., commonly used in many existing works. Secondly, we deal with the problem of tuning parameters of the similarity measure, an aspect which is quite important but usually omitted in existing works. For this purpose we exploit the results of statistical analysis of real-world XML data from Chapter 6, in particular XML schemes, and we show that the tuning problem can be viewed as a kind of constraints optimization problem and thus can be solved using corresponding approaches. For this purpose we exploit and compare two approaches – genetic algorithms and simulated annealing. Using further experiments we show that with an appropriate tuning the similarity measure can be much precise than a common "reasonable" setting usually used.

Most of the contents of this chapter has been published in the following paper:

Mlýnková, I.: Evaluation of XML Schema Fragments Similarity Based on Properties of Real Data. (Note: Paper under review process)

A possible optimization of XML-based methods can be found in exploitation of similarity of XML data and matching of XML patterns. In general it enables to manage similar XML data in a similar manner or to extend approaches known for a particular type of XML data to the whole set of

similar ones.

The most common area of exploitation of data similarity (not only for the XML case) are storage strategies based on the idea of *clustering* XML documents or XML schemes (e.g. [30]). They focus on storing structurally similar data in a similar way or "close" to each other to enable fast retrieval and to reduce processing of the whole set of stored data to their relevant subset. Another large area covers so-called *dissemination-based applications* (e.g. [20]), i.e. applications which timely distribute data from the underlying data sources to a set of customers according to user-defined profiles. These systems use a kind of approximate evaluation, since the user expects that the resulting data conform to the specified profile up to particular similarity threshold. Another set of similarity-based techniques consists of so-called *data integration systems*, or, when concerning directly XML schemes, *schema integration systems* (e.g. [48]). They enable to provide a user with a uniform view of the data coming from different sources and thus having different structure, identifiers, data types, etc. Hence such system must be able to analyze the source data and to find corresponding similarities. And there are also other, not such obvious examples of similarity exploitation [60], such as, e.g., data warehousing (e.g. [58]) which needs to transform the data from source format to the warehouse format, e-commerce where message translation between various formats is necessary (e.g. [31]), semantic query processing (e.g. [63]), etc.

But despite the fact that the amount of existing similarity-based approaches is significant, there is still a space for improvements and new ways of similarity exploitation. In this chapter we propose a similarity measure designed primarily for the purpose of enhancing of user-driven XML-to-relational storage strategies. But the key ideas we are proposing can be simply extended to be used to any appropriate similarity-based problem.

5.1 Related Work

As mentioned above, the number of existing works in the area of XML data similarity evaluation is nontrivial. We can search for similarity among XML documents, XML schemes, or between the two groups. We can distinguish several levels of similarity that can be taken into account during the search process, such as 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 value constraints), etc. Or, we can require different precisions of the similarity evaluation depending on the target application.

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. [57]) 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. [74]). 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. [25]) and techniques which measure the closest distance between D and "all" documents valid against S (e.g. [56]). 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, equality of data types and structure, schema instances, thesauri, previous results, etc. (e.g. [45] [33] [68])

In our case for choosing the best possible XML-to-relational storage strategy the key information lies in structural analysis of XML data. Thus, if we want to exploit similarity of XML data, also the corresponding measure should focus especially on structural level. And since the most common source of structural information are XML schemes, also the corresponding similarity should be evaluated primarily on schema level. In this area the key emphasis is currently put on the semantic similarity of the given schema fragments reflecting the requirements of corresponding applications (such as schema-integration systems, dissemination-based systems, etc.). But for the purpose of XML-to-relational storage strategies such techniques are inappropriate.

5.2 Proposed Similarity Evaluation

As mentioned in Chapter 4, the proposed similarity measure $sim(f_x, f_y) \in [0, 1]$ expressing similarity of two fragments f_x and f_y of schema S, where 1 represents strong similarity and 0 strong dissimilarity, is based on a similar idea as most of the existing works. It exploits a number of supplemental matchers $m_1, m_2, ..., m_p$ (see Definition 24), i.e. functions which evaluate similarity of a particular feature of the given schema fragments, such as, e.g., similarity of number of nodes of f_x and f_y, similarity of their depths,

similarity of their contents, etc. Then the partial results are combined into the resulting composite similarity value m_{comp} (see Definition 25 and 26). The most common and verified [33] way of composition is usually a kind of weighted sum.

In this chapter we focus on several key aspects of the similarity measure. Firstly, we deal with the problem of stating the partial matchers which should focus mainly on structural aspects of the given schema fragments and especially constructs which influence the efficiency of database processing. Thus the similarity should not primarily rely, e.g., on semantics of used identifiers. Furthermore, since our approach assumes that a user can annotate any kind of a schema fragment and a similar fragment can be detected anywhere in the schema, it should not rely also on context of the evaluated fragments. And last but not least, the similarity evaluation strategies have to cope with the problem of setting various parameters, especially weights of matchers within the composite measure. The setting process is usually omitted in current works (i.e. the weights are set without any argumentation), or a kind of a machine-learning technique is exploited.

With regard to these observations we have decided to exploit the knowledge and results of statistical analysis of real XML data collections, in particular the part considering XML schemes. The data parameters that were analyzed describe a given schema fragment quite precisely and, at the same time, the results for a representative sample of real XML schemes can be used for tuning parameters of the similarity measure.

5.2.1 Matchers and Composite Measure

For definition of matchers m_1, m_2, ..., m_p we exploit most of the XML data characteristics defined and evaluated in the analysis in Chapter 6. Since we want to describe the structure of the schema fragments as precisely as possible, the amount of characteristics is nontrivial. On the other hand, at this stage the versatility of the approach becomes evident, since in general any kind of matchers can be used depending on the purpose and requirements of corresponding application.

According to the scope the used characteristics can be divided into the following groups:

- *root* – characteristics of root node of the fragment:

  - type of content, i.e. empty, text, element, mixed, trivial, or unordered,
  - fan-outs, i.e. element, attribute, simple, minimum, and maximum,
  - type of the node, i.e. DNA pattern or relational pattern, and
  - type of the recursion, i.e. trivial, linear, pure, or general.

- *subtree* – characteristics of the whole fragment:
  - basic, i.e. number of elements, number of attributes, number of mixed contents, number of empty contents, maximum and average depth, minimum and maximum element fan-out,
  - XML Schema-like, i.e. number of unordered contents, default values, fixed values, wildcards, `ID` types, `IDREF(S)` types, `unique`, `key`, and `keyref` nodes,
  - recursive, i.e. number of recursive elements, number of particular types of recursion,
  - depths, i.e. minimum, maximum, average, and unbounded,
  - fan-outs, i.e. element, simple, minimum, and maximum,
  - mixed contents, i.e. depths, fan-outs, simple fan-outs,
  - DNA patterns, i.e. depths, fan-outs, simple fan-outs, and
  - relational patterns, i.e. fan-outs, simple fan-outs.

- *level* – characteristics of each level of the fragment: number of elements, attributes, text nodes, mixed contents, DNA patterns, and relational patterns, fan-outs and simple fan-outs.

Since each matcher should evaluate similarity of a particular characteristic of the given schema fragments f_x and f_y, we need to transform the resulting values (e.g. types of the root nodes, numbers of elements, etc.) to interval $[0, 1]$. In case of root characteristics we distinguish two cases – feature matchers and single-value matchers. *Feature matchers* express the (in)equality of the value of i-th feature fea_i (e.g. type of the node, content, or recursion):

$$m_i^{fea}(f_x, f_y) = \begin{cases} 1 & fea_i(f_x) = fea_i(f_y) \\ 0 & otherwise \end{cases} \tag{5.1}$$

They are combined using a weighted sum into a *composite feature matcher*:

$$m^{fea}(f_x, f_y) = \sum_{i=1}^{n} m_i^{fea}(f_x, f_y) \cdot w_i^{fea} \tag{5.2}$$

where weights $w_i^{fea} \in [0, 1]$, $\sum_{i=1}^{n} w_i^{fea} = 1$, and n is the number of feature matchers.

Single-value matchers express the difference between the value of j-th single-value characteristic $value_j$ (e.g. element or attribute fan-out):

$$m_j^{single}(f_x, f_y) = \frac{1}{|value_j(f_x) - value_j(f_y)| + 1} \tag{5.3}$$

As for the subtree characteristics we distinguish two cases too. In case of single-valued characteristics (i.e. basic, XML Schema-like, and recursive) we also use the single-value matchers (see equation 5.3) which are within the subsets composed into *composite single-value matchers* m^{single}, again using a weighted sum. The situation in case of multi-valued characteristics (e.g. list of possible depths of the fragment) is a bit complicated – it requires similarity evaluation of two lists of values of arbitrary, distinct lengths. Therefore we first supply the shorter list with zero values and sort the lists in decreasing order. Then we use so-called *multi-valued matchers* which express the similarity of a j-th sorted sequence s_j:

$$m_j^{multi}(f_x, f_y) = \frac{\sum_{k=1}^{m} \frac{1}{|s_j(f_x)[k] - s_j(f_y)[k]| + 1}}{m} \tag{5.4}$$

where m is the length of the sequences and $seq_j(.)[k]$ expresses the k-th member of the sequence.

For level characteristics we use so-called *level matchers* which compose the results of single-valued (see equation 5.3) or multi-valued (see equation 5.4) matchers at particular levels and decrease their weight with the growing level:

$$m_j^{lev}(f_x, f_y) = \sum_{k=1}^{l} m_j^{single/multi}(f_x, f_y) \cdot (\frac{1}{2})^k \tag{5.5}$$

where l is the maximum of number of levels of f_x and f_y (assuming that the shallower one is again supplied with zero values).

Finally, the resulting composite function m_{comp} is expressed as a weighted sum of all the matchers.

As it is obvious, the resulting composite similarity expresses the similarity of the given schema fragments with regard to the selected matchers, each having its particular weight. Thus there remains the problem of tuning the weights which highly influences its precision.

5.2.2 Tuning of the Weights

In existing works we can distinguish two approaches – the parameters are set either without any argumentation (or on the basis of authors experience whose more detailed description is usually omitted) or a machine-learning strategy is exploited. In the latter case the corresponding system is usually provided with a set of sample situations (e.g. pairs of data fragments and their similarity) and the system than exploits this knowledge in the evaluation process.

In our approach we use the "golden mean" – we exploit the results from the analysis of real-world XML schemes (see Chapter 6) and we set the parameters on the basis of the results. The basic idea is relatively simple: We use the same 98 real-world XML schemes divided into database (`dat`), document (`doc`), exchange (`ex`), report (`rep`), and research (`res`) category. Their basic characteristics can be seen in Tables 6.3 and 6.4. We prepare sample patterns of real schema fragments, such as data-centric fragments, document-centric fragments, unordered elements, relational patterns, recursive elements (of all the four types), DNA patterns, etc., whose representation is in the particular categories known. Using a search algorithm we compute the number of occurrences of similar fragments within the schema categories and tune the parameters of the similarity measure so that the results correspond to the results of analysis of the real-world data.

Note that this is the second stage where the algorithm can be modified to any purpose. In general it is possible to use any relevant information, i.e. knowledge of characteristics of any sample set of data. We have used the results of our analysis since the real-world sample is nontrivial and the data were collected so that they cover many possible areas where XML data are exploited.

Theoretical View of the Tuning Problem

In general the tuning problem and its proposed solution can be described as follows: Let $c_1, c_2, ..., c_K$ denote the categories of schemes, $p_1, p_2, ..., p_P$ the sample patterns, and $(M^{rep}_{i,j})_{K \times P}$ the *representation matrix* which contains *real-world representation* of pattern p_j in category c_i, i.e. results of the statistics. Next let us have a search algorithm with parameters $par_1, par_2, ..., par_R$, where $\forall i : par_i \in [0, 1]$ and some subsets of the parameters have to fulfill particular constraints, such as, e.g., the sum of subset of parameters which correspond to weights of a single weighted sum must be equal to 1. With the given setting of parameters the algorithm returns *calculated representation* $rep_{i,j}$ of pattern p_j in category c_i. The aim is to find the optimal setting of parameters $par_1, par_2, ..., par_R$, i.e. the setting where the sum of deviations of calculated and real-world representations

$$\sum_{i=1}^{K} \sum_{j=1}^{P} |M^{rep}[i, j] - rep_{i,j}| \tag{5.6}$$

is minimal. This task is obviously a kind of a classical *constraints optimization problem (COP)* – a problem of finding a solution in a *feasible region* (i.e. a set of all possible solutions), where the value of *objective function* (i.e. a function which determines the quality of a solution) is optimal and the solution satisfies the given criteria. In our case:

- the feasible region contains all possible settings of parameters par_1, par_2, ..., par_R corresponding to weights of weighted sums (defined in Chapter 5.2.1),
- the objective function evaluates the sum 5.6, and
- the criteria of the solution are the above described constraints.

The problem is that since the parameters $par_1, par_2, ..., par_R$ are in general real values from $[0, 1]$, the feasible region is theoretically infinite. But under a closer investigation we can see that the real-world case is much simpler than the theoretical problem. Firstly, we can restrict possible values of parameters par_1, par_2, ..., par_R to a certain precision, i.e. the infinite feasible region can be reduced to a reasonable size. Furthermore, for our purpose we do not need the optimal solution, but a reasonably good suboptimum, since the

algorithm is expected to search for similar schema fragments, not exactly the given ones. And last but not least, as the evaluation of the objective function requires similarity evaluation of all the patterns $p_1, p_2, ..., p_P$ and all schema fragments in categories $c_1, c_2, ..., c_K$, we need to minimize the amount of evaluations.

For searching the suboptimum we exploit and compare a slight modification of two approaches for searching a suboptimal solution of an optimization problem – global search heuristics called *genetic algorithms* and *simulated annealing*. They enable to find a reasonable setting following the given requirements and, at the same time, influence the number of expensive evaluations.

Genetic Algorithms

Genetic algorithms (GA) [37] are a part of evolutionary algorithms which are inspired by observations of evolution biology. Their idea is based on iterative improving of (usually) randomly generated *initial population* P_0 of individuals using two key operations, simulations of natural processes – *crossover* and *mutation*.

As depicted by Algorithm 6, at i-th iteration the *fitness* f_{fit}, i.e. the quality, of every individual of population P_i is evaluated, multiple individuals are selected on the basis of their fitness, and modified, i.e. crossed over and mutated to form a new population P_{i+1}. Operation crossover creates a new offspring crossing over two individuals by exchanging their portions. Operation mutation creates a new individual by changing attributes of an existing one. Both the operations are performed with a given probability P_{cross} and P_{mut} which influence the speed of convergence to the suboptimal solution. The algorithm terminates either if satisfactory *fitness level* F_{min} has been reached in population P_i or after N iterations.

In our case a single individual of a population corresponds to a single possible setting of parameters $par_1, par_2, ..., par_R$ and the fitness function evaluates the inverse value of sum 5.6. The initial population is generated randomly or a set of reasonable settings can be used. And finally, operations crossover and mutation are slightly modified to ensure that the subsets of parameters corresponding to a single weighted sum still fulfill the previously described conditions.

Algorithm 6 Genetic Algorithm (GA)

Input: f_{fit}, F_{min}, N
Output: individual with the maximum fitness

```
1: i ← 0
2: P_i ← initial population
3: evaluate individuals in P_i using f_fit
4: while i ⩽ N ∧ F_min is not reached in P_i do
5:     i ← i + 1
6:     P_i ← best individuals from P_{i−1}
7:     crossover(P_i)
8:     mutate(P_i)
9:     evaluate individuals in P_i using f_fit
10: end while
11: return I ∈ P_i s.t. f_fit(I) is maximum
```

Simulated Annealing

The idea of *simulated annealing (SA)* [38] is also inspired by natural processes, in this case the way a metal cools and freezes into crystalline structure, where controlled cooling increases size of the crystals and thus reduces defects. SA algorithm is also iterative and based on exploitation of randomly generated solutions.

As depicted by Algorithm 7, the SA algorithm starts with the *initial state* s_0 which is iteratively improved. The quality of a state s_x is evaluated using its *energy* $E(s_x)$ which needs to be minimized. At i-th iteration the current state s_i is replaced with a random "nearby" state s_{i+1} whose choice depends on a global parameter T called *temperature* which is gradually decreased (usually by fixed factor $\alpha < 1$) during the process. The probability P_{mov} of moving from state s_i to s_{i+1} is expressed as a function of T, $E(s_i)$, and $E(s_{i+1})$:

$$P_{mov} = \begin{cases} 1 & E(s_i) > E(s_{i+1}) \\ exp(\frac{E(s_i)-E(s_{i+1})}{T}) & otherwise \end{cases} \tag{5.7}$$

The algorithm terminates either after a certain number of iterations N or if a state with satisfactory energy E_{min} is reached.

The main advantage of SA in comparison with GA is its ability to avoid trapping at local optimum. The reason is that SA does not accept only

Algorithm 7 Simulated Annealing (SA)

Input: E, E_{min}, N
Output: state with minimum energy E

1: $i \leftarrow 0$
2: $s_0 \leftarrow$ initial state
3: $s_{opt} \leftarrow s_0$
4: **while** $i \leqslant N \wedge E_{min} < E(s_{opt})$ **do**
5: $\quad i \leftarrow i + 1$
6: $\quad s_i \leftarrow$ a random neighbor of s_{i-1}
7: $\quad$ **if** $E(s_i) < E(s_{opt})$ **then**
8: $\quad\quad s_{opt} \leftarrow s_i$
9: $\quad$ **end if**
10: $\quad$ **if** $\neg$ move $(E(s_{i-1}), E(s_i), T)$ **then**
11: $\quad\quad s_i \leftarrow s_{i-1}$
12: $\quad$ **end if**
13: $\quad$ decrease(T)
14: **end while**
15: **return** s_{opt}

states which improve the current optimum, but also some of those which can (temporarily) worsen it. The probability P_{mov} and temperature T ensure that at the beginning the state changes almost arbitrarily (within the adjacent states) but the changes decrease as T goes to zero.

In our case each state represents a single setting of parameters par_1, par_2, ..., par_R and the energy E evaluates sum 5.6. For the initial state can be again used either a randomly generated setting or any known reasonable setting. The neighboring states are defined by a modification of mutation of GA, where only a single parameter is randomly changed (and the others are recomputed to fulfill conditions of weighted sums).

5.3 Experimental Tests

The experimental tests we have performed to analyze the proposed ideas and approaches can be divided into two parts. Firstly, we analyze the tuning process, i.e. we compare the two algorithms for searching a better candidate for the weights and the achieved results. And secondly, we analyze the quality

of the tuned similarity measure.

5.3.1 Tuning Process

For experimental testing of GA and SA algorithms we first need to tune the key parameters of both the algorithms.

As for the GA algorithm, we need to tune the probabilities P_{cross} and P_{mut} which influence the speed of convergence to the suboptimum. With fixed value of P_{mut} and maximum number of iterations $N = 30$ we have performed number of tests for setting the value of P_{cross}. The results are depicted by Figures 5.1 and 5.2 containing average, median, and minimum values for values of $P_{cross} \in [0.1, 0.9]$. Figure 5.1 depicts at which iteration the suboptimal value was reached, Figure 5.2 depicts the resulting suboptimal value of sum 5.6. As we can see, since both the average and median iteration for reaching the suboptimum occur mostly between 15 and 20, the maximum number of iterations does not need to be much higher. Considering only the reached values (Figure 5.2) the best candidates for P_{cross} are 0.1, 0.4, 0.5, 0.7 and 0.9. But together with the results from Figure 5.1 we can say that the optimal value of P_{cross} occurs between 0.4 and 0.5.

Similarly, with fixed value of $P_{cross} = 0.5$ and the same maximum number of iterations $N = 30$ the process of tuning of parameter P_{mut} is depicted in Figures 5.3 and 5.4 containing values with the same meaning. Unfortunately, these results are more ambiguous than in the previous case. Thus we have analyzed the particular results and from the possible candidates we have selected 0.16 as the best compromise value of P_{mut}.

As for the SA algorithm, we need to perform the same tuning for parameter T and the threshold of P_{mov}. The setting of T requires quite a lot of user involvement since the value must conform to numerous requirements to achieve a reasonable value [38]. Mainly, it should correspond to the estimated deviation of $E(s_i)$ and $E(s_{i+1})$, it must ensure that the algorithm moves to a state s_{i+1} even if $E(s_{i+1}) > E(s_i)$ (but the probability P_{mov} is high enough), the declination of T should be reasonable enough to exploit the main idea of the approach, etc. Thus this parameter was set rather semiautomatically with regard to balanced conformation to the requirements. With fixed value of T and maximum number of iterations $N = 80$ the value of P_{mov} seems to be the best around 0.7 considering the number of iterations and between 0.4 and 0.5 considering the resulting values, as depicted by Figures 5.5 and 5.6 respectively. After more detailed analysis of the results, we have set P_{mov} to

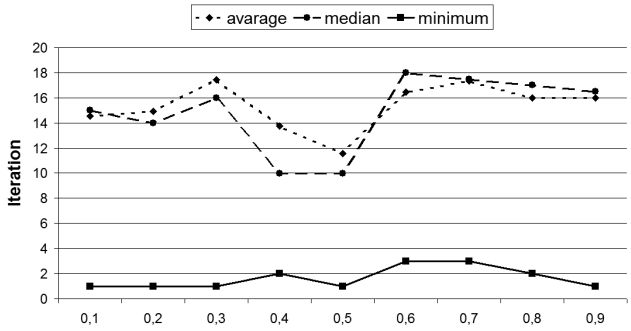


Figure 5.1: Tuning of parameter P_{cross} – number of iterations

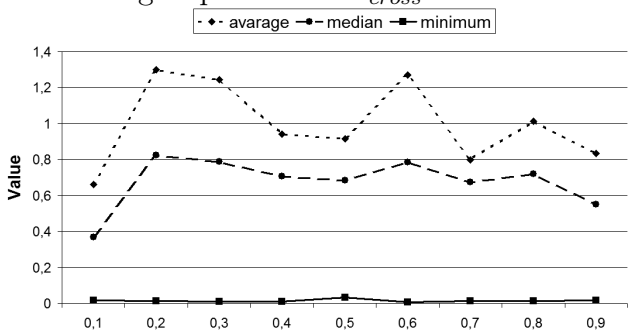


Figure 5.2: Tuning of parameter P_{cross} – values of sum 5.6

0.7.

With the current setting of both the algorithms we can now analyze their behavior. Firstly, we are interested in the quality of the achieved suboptimum, in particular in comparison with the typical reasonable setting of weights so that the composite measure returns the average similarity. Table 5.1 overviews the quality of the suboptimums expressed using the result of sum 5.6 for both the GA and SA algorithms which were evaluated either starting with random population P_0/state s_0 or with setting to the average-producing weights (denoted as `avg`). As we can see in both the cases the results are much better when we start with a reasonable, verified setting than with a random one. If we compare the quality of `avg` with the achieved suboptimums, we can see that using both the algorithms can found much better candidates for setting the weights.

Comparing the two algorithms together we are interested in the amount of expensive evaluations of fitness/energy (i.e. the sum 5.6), in particular their efficiency and the quality of the reached suboptimum. As for the quality we

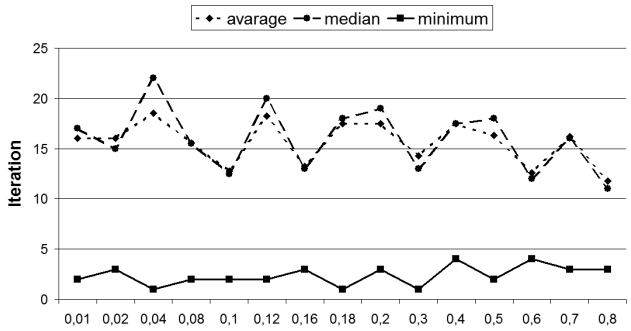


Figure 5.3: Tuning of parameter P_{mut} – number of iterations

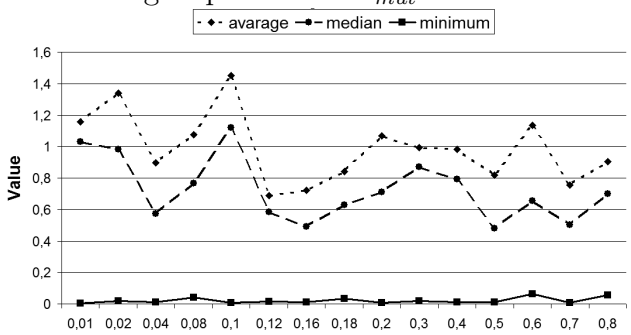


Figure 5.4: Tuning of parameter P_{mut} – values of sum 5.6

can again refer to Table 5.1, where we can see that though the values of the two algorithms do not differ too much, the GA algorithm performs better in all the cases.

The analysis of number of iterations firstly requires a brief discussion: In case of GA the number of evaluations of sum 5.6 seems to be given by the product of number of iterations and number of individuals in a population. In case of SA it is given by the number of iterations, since at each iteration only a single state is evaluated. But due to the properties of GA all the individuals of a population can be evaluated concurrently avoiding repetitions of expensive preprocessing of the graph and supplemental calculations (e.g. number of nodes, depths, types of nodes, etc.). Thus in fact also in this case the number of evaluations rather corresponds to number of iterations of the algorithm.

The resulting numbers of iterations necessary for reaching the suboptimums from Table 5.1 are depicted in Table 5.2. As can be seen not only has the GA algorithm undoubtedly better results than SA, but also the number

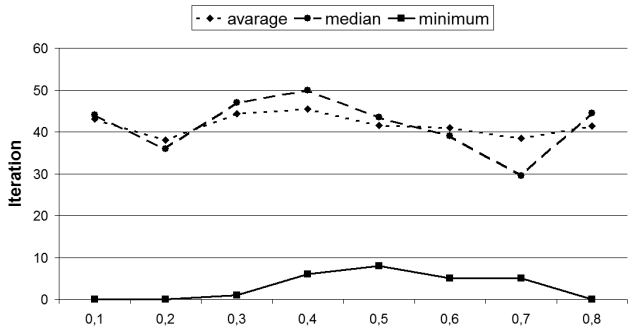


Figure 5.5: Tuning of parameter P_{mov} – number of iterations

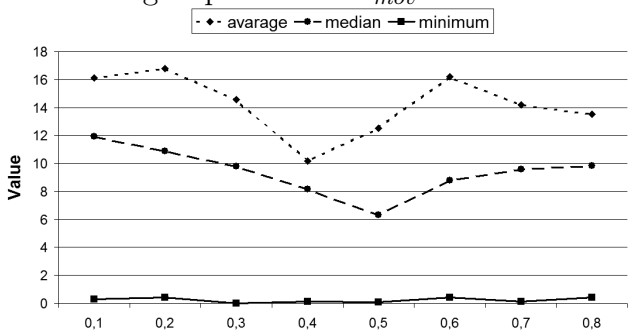


Figure 5.6: Tuning of parameter P_{mov} – values of sum 5.6

of iterations is lower due to its ability to evaluate a population of candidates at each iteration instead of a single one. It is quite probable that with a higher number of iterations the SA algorithm would perform better, but the duration of such evaluation would be unacceptable as it cannot be performed concurrently for more candidates. Thus though the SA algorithm is able to cope with local optimums [38], for our purpose seems to be better to use an approach which is able to reach the optimum as soon as possible, as it is in case of GA assured using a population of candidates.

5.3.2 Similarity Measure

At this stage we have a similarity measure whose parameters are tuned according to the knowledge of structure of real-world data. But the question is how good such tuning is. As we have mentioned, the existing works rather focus on semantic similarity of XML schema fragments and thus comparison of our approach with any of them would be misleading. On the other

Characteristic		Result of sum 5.6			
		Minimum	**Average**	**Median**	**Maximum**
P_0 (GA)	random	0,013	1,176	0,673	3,959
	`avg`	0,001	0,652	0,463	3,441
s_0 (SA)	random	0,082	17,318	11,764	55,719
	`avg`	0,061	9,412	6,595	40,519
Arithmetic mean		482,113			

Table 5.1: Quality of the achieved suboptimums

Characteristic		Number of iterations			
		Minimum	**Average**	**Median**	**Maximum**
P_0 (GA)	random	1	17,2	19	30
	`avg`	5	20,9	22,5	30
s_0 (SA)	random	8	39,8	38	80
	`avg`	2	38.7	37	80

Table 5.2: Efficiency of achieving the suboptimum

hand, we can compare the tuning with the usually used reasonable setting to the average-producing weights. From Table 5.1 we can see that in terms of sum 5.6 the reached settings are much better than in the average-producing case. But such results are not very convincing in general. Thus for the purpose of evaluation of quality of the two similarity measures (which we further denote as `SimTuned` and `SimAvg`) we use the approach introduced in [33]. It is based on the idea of comparing results of an algorithm with results of manual processing assuming that the manually achieved results form the optimum. Let R be the set of manually determined matches, i.e. in our case schema fragments similar to the given schema pattern, and P the set of matches determined by the algorithm. Then I denotes the set of *true positives*, i.e. matches correctly identified by the algorithm, $F = P \backslash I$ denotes *false matches*, i.e. matches incorrectly identified by the algorithm, and $M = R \backslash I$ denotes *false negatives*, i.e. matches not identified by the algorithm. On the basis of these characteristics, the following quality measures can be computed:

- $Precision = \frac{|I|}{|P|} = \frac{|I|}{|I|+|F|}$ estimates the reliability of the similarity measure,
- $Recall = \frac{|I|}{|R|}$ specifies the share of real matches that is found, and

- *Overall* $= 1 - \frac{|F|+|M|}{|R|} = \frac{|I|-|F|}{|R|}$ represents a combined measure which represents the post-match effort necessary to remove false and add missed matches.

In the ideal case $I = P = R$, $F = M = \emptyset$, and the measures reach their highest values Precision = Recall = Overall = 1. On the other hand, the lower the values are (whereas the Overall can have even negative values), the least precise the similarity measure is.

For the purpose of the evaluation we have selected 5 XML schemes representing each of the 5 schema categories and prepared a sample set of 10 data-centric and 10 document-centric schema patterns. For each of the schema representatives we have manually identified the set R of data-centric and document-centric fragments. Then we have performed searching for fragments similar to the schema patterns using both `SimAvg` and `SimTuned`, i.e. we have performed $5 \times (10 + 10)$ experiments for each of them, and determined the sets P and I. Finally, within the categories we have computed average values of the results (i.e. average $|P|$ and $|I|$) and then the resulting values of Precision, Recall, and Overall.

As can be seen from Figure 5.7 the `SimAvg` approach is apparently much worse similarity measure than `SimTuned` within all the categories. Secondly, it is evident (and natural) that the quality of both the measures is correlated with the number of source schemes within the categories, i.e. the amount of source information we had for tuning the weights. The best results can be found for categories `dat`, `doc`, and `ex` since the amount of corresponding schemes highly exceeds the amount in the other two (see Table 6.2).

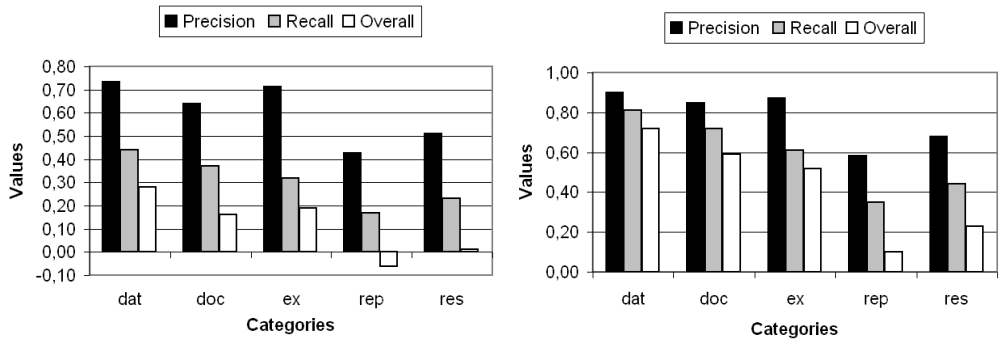


Figure 5.7: Precision, Recall, and Overall for `SimAvg` and `SimTuned`

If we consider the Precision and Recall parameters together (since they influence each other and thus cannot be evaluated separately, otherwise the results can be misleading), we can see that in the first three categories the reliability as well as the share of real matchers found exceeded 60%. It is not as good result as in case of [33], where for the best identified similarity measures the values often exceeded 75%, but none of the evaluated similarity measures focussed on structural similarity as precisely as in our case and, of course, the match tasks were quite different.

Considering the Overall parameter the worst results are again in case of `rep` and `res` categories, whereas in case of `SimAvg` and `rep` category the value is even negative. This denotes that the number of false positives exceeds the number of true positives and such measure is almost useless, since the post-match effort is too high.

5.4 Conclusion

The experiments show that with the proper tuning of weights based on reliable information on representative sample data, the corresponding similarity measure has much better characteristics than the commonly used average-producing ones. In addition, the idea can be used for any type similarity measure (i.e. measure focussing not only on structural similarity) and any type of relevant tuning data.

Last but not least, note that from another point of view also the semantics of element/attribute names can be important. In our approach we assume that for the purpose of XML-to-relational storage strategies the key aspect is structural similarity of the data. This is definitely true but from another point of view we could assume that not only structurally similar but also semantically similar schema fragments should be stored in a similar way. In another words we could expect that the user will work similarly with semantically similar fragments.

Chapter 6

Statistical Analysis of Real-World XML Data

In this chapter we describe a statistical analysis of existing real-world XML data, in particular their structure and real complexity. The results were used primarily in the tuning process of similarity measure in Chapter 5 and experiments in Chapter 4, but the analysis in general brings several important findings and conclusions. For its purpose we have gathered more than 20GB of real XML collections and implemented a robust automatic analyzer. The analysis considers existing papers on similar topics, trying to confirm or refute their observations as well as to bring new findings. It focuses on frequent, but often ignored XML constructs (such as mixed content or recursion) and relationship between schemes and their instances.

Most of the contents of this chapter has been published in the following papers:

Mlýnková, I. – Toman, K. – Pokorný, J.: Statistical Analysis of Real XML Data Collections. COMAD '06: Proceedings of the 13th International Conference on Management of Data, pages 20 – 31, New Delhi, India, December 2006. Tata McGraw-Hill Publishing Company Limited, 2006. ISBN 0-07-063374-6. (Note: The Best Student Paper Award)

Toman, K. – Mlýnková, I.: XML Data – The Current State of Affairs. Proceedings of XML Prague '06 conference, pages 87 – 102, Prague, Czech Republic, June 2006. (Note: An invited talk)

When considering the existing works on XML data processing in more detail, we can distinguish two situations. On one hand, there is a group of general

techniques that take into account all possible features of input XML data. This idea is obviously correct, but the problem is that the XML standards were proposed in full possible generality so future users can choose what suits them most. Nevertheless, the real XML data are usually not so "rich", thus the effort spent on every possible feature is mostly useless. It can even be harmful in terms of efficiency.

On the other hand, there are techniques that somehow do restrict features of the input XML data. For them it is natural to expect inefficiencies to occur only when the given data do not correspond to these restrictions. The problem is that such restrictions do not result from features of real XML data, but they are often caused by limitations of a particular technique, complexity of such solution, irregularities, etc.

We can naturally pose two apparent questions:

1. Is it necessary to take into account a feature that will be used minimally or will not be used at all?

2. If so, what are these features?

The answer for the first question obviously depends on the particular situation. The second one is the main topic of this chapter.

6.1 Related Work

So far only a few papers have focused on analysis of real XML data. They analyze either the structure of DTDs, the structure of XSDs, or the structure of XML documents (regardless their schema). The sample data usually differ.

For the first time the analysis of the structure of DTDs had probably occurred in paper [65] and it was further extended in papers [29] and [40]. They focused especially on the number of (root) elements and attributes, the depth of content models, the usage of mixed content, `ID`s/`IDREF`s, and attribute "decorations" (i.e. `implied`, `required`, and `fixed`), non-determinism, and ambiguity. Side aim of the papers was a discussion of shortcomings of DTDs. The most important findings are that real content models are quite simple (the depth is always less than 10), the number of non-linear recursive elements (see Definition 30) is high (they occur in 58% of all DTDs), the number of hubs is significant, and that `ID`s/`IDREF`s are not used frequently.

With the arrival of XML Schema, a natural question has arisen: Which of the extra features of XML Schema not allowed in DTD are used in practice? Paper [26] is trying to answer it using statistical analysis of real XML schemes. The most exploited features seem to be restriction of simple types (found in 73% of XSDs), extension of complex types (37%), and namespaces (22%). The first finding reflects the lack of types in DTD, the second one confirms the naturalness of object-oriented approach, whereas the last one probably results from mutual modular usage of XSDs. The other features are used minimally or not used at all. The concluding finding is that 85% of XSDs define so called *local tree languages* [54], i.e. languages that can be defined by DTDs as well. Paper [46], that also focuses directly on structural analysis of XSDs, defines 11 metrics and two formulae that use the metrics to compute complexity and quality indices of XSDs. Unfortunately, there is only one XSD example for which the statistics were computed.

Paper [47] analyses the structure of XML documents directly, regardless eventually existing schema. The statistics are divided into two groups – statistics about the XML Web (e.g. clustering of the source web sites by zones and geographical regions, the number and volume of documents per zone, the number of DTD/XSD references, etc.) and statistics about the XML documents (e.g. the size and depth, the amount of markup and mixed-content elements, fan-out, recursion, etc.). The most interesting findings of the research are that the structural information always dominates the size of documents, both mixed-content elements (found in 72% of documents) and recursion (found in 15% of documents) are important, and that documents are quite shallow (they have always fewer than 8 levels on average).

In this chapter we take up work initiated in the existing articles. We focus on analysis of XML data aspects which are important for efficient data processing, while we omit, e.g., the source of XML data collections or secondary XML items such as name space references. In general, our analysis focuses on aspects which influence the structural complexity of XML data or carry additional important information, while it ignores items that are rather relevant to semantic web [16] or can be even qualified as "syntactic sugar".

6.2 Sample XML Data Collections

We have collected a huge amount of XML documents and their DTDs/XSDs. In contrast to existing papers, the XML collections were not only collected

automatically using a crawler, but also manually from sources offering their data natively in XML format (e.g. government sites, open document repositories, web site XML exports, etc.), Internet catalogues, and semantic web resources. The respective schemes for data sets were often searched out later in separate because they had been missing in the original sources. Then the collections were categorized and the duplicate documents identified by a simple hashing algorithm (disregarding white spaces), and subsequently removed. Also computer-generated or random-content XML documents were eliminated.

The reason for using more reliable and/or categorized sources is that automatic crawling of XML documents generates a set of documents that are "unnatural" and often contain only trivial data which cause misleading results. For example paper [47] mentions that the set of sample data (which were crawled automatically) contains almost 2000 of documents with depth $\leqslant 1$, i.e. documents containing a single element with empty or text content. Our purpose was to collect a representative set of currently used XML collections from various spheres of human activities. We have included data which are used for testing XML processing methods (e.g. Shakespeare's plays [1], XMark [2], Inex [3]), representatives of standard XML schemes (e.g. XHTML [18], SVG [19], RDF [24], DocBook [10]), sample database exports (e.g. FreeDB [4], IMDb [5]), well-known types of documents (e.g. OpenOffice documents [13]), randomly crawled XML data (Medical Subject Headings [6], novels in XML [7], RNAdb [8]), etc.

6.2.1 Preprocessing

Authors of most existing papers complain of a large number of errors in the collected sample data. Unfortunately, we can only confirm the claim. The overwhelming majority of the collected data contained various types of serious errors. XML documents were not even well-formed and more than a half of the well-formed ones contained invalid data. Similar problems were found in case of DTDs and XSDs.

Contrary to previous papers we have not decided to discard the data. Most of the errors (e.g. bad encoding, missing end tags, missing elements, unescaped special characters, wrong usage of namespaces, etc.) were detected using Xerces Java parser [59] or various auxiliary analyzers and semi-automatically corrected. Most of the corrections had to be done manually though.

6.2.2 Accessibility of the Data

Unfortunately, we cannot release the resulting set of corrected XML data collections for download, since most of them are not free or underlie to Copyright Act and do not allow redistribution. The complete listing of the original data, number of documents per collection, DTD/XSD existence, and their sources can be found in [52]. (Naturally, we cannot be responsible for their current validity.)

6.2.3 General Metrics and Classifications

First of all, we have computed statistics that describe the sample XML data in general. The overview of these statistics is listed in Table 6.1.

Statistics	**Results**
Number of XML documents	16,534
Number of XML collections	133
Number of DTDs/XSDs	98
Total size of documents (MB)	20,756
Minimum size of a document (B)	61
Maximum size of a document (MB)	1,971
Average size of a document (MB)	1.3
Median size of a document (kB)	10
Sample variation (MB)	433.8
Documents with DTD (%)	74.6
Documents with XSD (%)	38.2
Documents without DTD/XSD (%)	7.4

Table 6.1: General statistics for XML data

The sizes of XML documents vary strongly (from 61B to 1,971MB), nevertheless both the average size (1.3MB) and median size (10kB) seems to be "natural". Another not surprising finding is that a considerable percentage of documents (7.4%) still does not have any schema (although the ratio is better than in all mentioned existing works) and if so, the XML Schema language is for this purpose used even less (only for 38.2% of documents).[1] The positive results may be influenced by the fact that the gathered data were collected semi-automatically, not randomly.

[1]Some documents have both DTD and XSD, thus the sum is not 100%.

To avoid averaging the features of the whole data set, where the documents have nothing in common but having an XML format, we have categorized the data by the original sources and further grouped according to similar structure, contents, or approach used to describe the data. This way we have obtained a finer look into various types of XML data not neglecting the interesting differences among the selected categories, whereas we have avoided the extensive amount of similar results.

In the rest of the chapter we refer to the following categories[2]:

- *data-centric documents* (`dat`), i.e. documents designed for database processing (e.g. database exports, lists of employees, lists of IMDb movies and actors, etc.),
- *document-centric documents* (`doc`), i.e. documents which were designed for human reading (e.g. Shakespeare's plays, XHTML documents, novels in XML, DocBook documents, etc.),
- *documents for data exchange* (`ex`) (e.g. medical information on patients and illnesses, etc.),
- *reports* (`rep`), i.e. overviews or summaries of data (usually of database type),
- *research documents* (`res`), i.e. documents which contain special (scientific or technical) structures (e.g. protein sequences, DNA/RNA structures, NASA findings, etc.), and
- *semantic web documents* (`sem`), i.e. RDF documents.

The number and size of documents per each category is depicted in Table 6.2 and for better clarity also in pie charts in Figure 6.1. The complete assignment of XML collections into particular categories can be found in [52].

Considering the sizes of individual XML documents, the most homogenous category is the `doc` one. Surprisingly it is also the one with the least document size (61B). Apparently this is because most of these XML documents are written by hand or with the aid of an XML editor. The other

[2]Though these logical categories are not as strictly defined as the common data/document-centric partitioning, no XML document has been inserted into more than one category.

Statistics	dat	doc	ex	rep	res	sem
Number of XML documents	3,412	6,691	218	2,983	2,451	779
Number of XML collections	38	22	25	2	16	30
Number of DTDs/XSDs	31	18	38	4	7	0
Total size of documents (MB)	2,237	1,187	371	11,371	1,697	3,892
Min. size of document (B)	447	61	2,433	1,925	2,016	356
Max. size of document (MB)	1,242	16	134	96	684	1,971
Avg. size of document (kB)	672	182	1,744	3,903	709	5,116
Median size of document (kB)	3.8	13.4	5.7	1,574	6.9	45.0
Sample variation (MB)	510.7	0.6	154.4	61.8	352.3	5534.5
Documents with DTD (%)	99.7	93.7	100	0	99.8	0
Documents with XSD (%)	0	57.8	0	100	99.6	0
Schema-less documents (%)	0.3	6.3	0	0	0.2	100

Table 6.2: General statistics per category

extreme is the `sem` category which, despite the similar structure of documents, contains some very large documents (including the largest one with 1,971MB) as well as collections split into very small documents, depending heavily on the used tool and conventions.

Note that the percentage of usage of DTDs or XSDs is always either almost 0% or almost 100%, depending on the category. The reason probably comes from the higher reliability of the used sources and the chosen categorization. On the other hand, though a considerable portion of all documents used some kind of a standard schema (e.g. XHTML, DocBook, etc.), they were not 100% valid against it. Thus the schema could be used as a guide for human processing, but it would not be usable for computer validation.

6.3 Analyses and Results

There are two main parts of our observations. Firstly, we carry out analyses similar to previous studies and compare the results with the original ones. Secondly, we focus on new XML features with emphasis on frequently disregarded ones such as mixed content and recursion, their complexity, and corresponding classification.

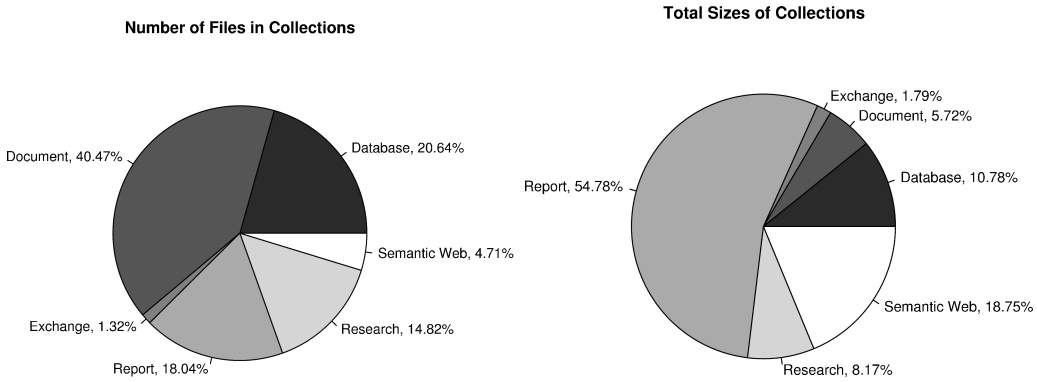


Figure 6.1: Number and size of documents per category

6.3.1 New Constructs

On the basis of our experience and manual preprocessing of the analyzed data we have defined several new constructs which describe XML documents and schemes with higher degree of detail. The constructs involve a new type of element content and related constructs, three types of element fan-out, four types of recursion, and two special types of elements called relational and DNA patterns. We have focused especially on simple patterns within the general XML constructs which can usually be represented by "ordinary" relational tables or, in case of recursion and mixed contents, which can be processed and stored simply, without the usual generalization.

Definition 28 *An element is* trivial *if it has an arbitrary amount of attributes and its content model* $\alpha = \epsilon \mid pcdata$.

A mixed-content element is simple *if each of its subelements is trivial. A mixed-content element that is not simple is called* complex.

A simple element fan-out *of an element* e *is the number of distinct trivial elements in its content model* α.

Definition 29 A minimum element fan-out *of element* e *is the minimum number of elements allowed by its content model* α.

A maximum element fan-out *of element* e *is the maximum number of elements allowed by content model* α.

An unbounded element fan-out *is a maximum element fan-out of* ∞.

Definition 30 *An element e is* trivially recursive *if it is recursive and e is the only element reachable from e and neither of its occurrences is enclosed by "*" or "+".*

An element e is linearly recursive *if it is recursive and e is the only recursive element reachable from e and neither of its occurrences is enclosed by "*" or "+".*

An element e is purely recursive *if it is recursive and e is the only recursive element reachable from e.*

An element that is recursive but not purely recursive is called a generally recursive *element.*

Definition 31 *A nonrecursive element e is called* a DNA pattern *if it is not mixed and its content model α consists of a nonzero amount of trivial elements and one nontrivial and nonrecursive element whose occurrence is not enclosed by "*" or "+". The nontrivial subelement is called* a degenerated branch.

A depth *of a DNA pattern e is the maximum depth of its degenerated branch.*

Definition 32 *A nonrecursive element e is called* a relational pattern *if it has an arbitrary amount of attributes and its content model $\alpha = (e_1, e_2, ..., e_n)*$ | $(e_1, e_2, ..., e_n)+$ | $(e_1|e_2|...|e_n)*$ | $(e_1|e_2|...|e_n)+$, where e_1, e_2, ..., e_n are trivial elements.*

A nonrecursive element e is called a shallow relational pattern *if it has an arbitrary amount of attributes and its content model $\alpha = f* \mid f+$, where f is a trivial element.*

6.3.2 Statistics and Results

In this chapter we discuss only the core findings and results; all results can be found in [52]. When possible, we also compare results of the analyses of XML documents with corresponding results of analyses of their XML schemes.[3] There are of course no schema results for `sem` category, since it contains no schema at all.

[3]In such case there are **Doc.** and **Sch.** abbreviations on the left-hand side of a table which identify the respective results.

Global Statistics

The first set of statistics we call *global*, since they consider overall properties of XML data. They involve the number of elements of various types (i.e. empty, text, mixed, recursive, etc.), the number of attributes, the text length in document, paths, and depths. For DTDs/XSDs the depths are counted for each root element used in the sample XML documents, for recursive elements we take into account the lowest level(s) and the infinite level.

Table 6.3 gives us the notion of limits of a "typical" XML document of each category. Each column contains the overall results of all documents disregarding exactly 5% of those ones with the extreme values. For the sake of completeness the extremes are listed in Table 6.4.

Statistics		**dat**	**doc**	**ex**	**rep**	**res**	**sem**
Max. num. of elements		402	4,085	37,502	309,379	427	112,942
Max. num. of attributes		9	1,675	5,182	37,815	129	37,996
Max. num. of empty elements		3	361	123	16,348	6	23,635
Max. num. of mixed elements		0	302	21	0	1	0
Max. num. of dist. el. names		81	48	58	388	44	144
Max. num. of rec. elements		0	3	2	0	0	0
Max. num. of distinct paths		79	96	67	312	30	143
Depth of document	Avg.	5	7	5	5	5	5
	Max.	5	13	9	6	7	6

Table 6.3: Global statistics for 95% XML documents

It is evident that most of the documents are constructed quite simply using only a very reasonable number of distinct element and attribute names (usually less than 150) which influences a similar number of distinct paths within each category. And even though the maximum number of elements in an XML tree is often huge, the number of distinct paths still remains several orders lower. This naturally corresponds with the average and maximum depths of XML documents which are very low (all under 13 disregarding the 5% of extreme values). Nevertheless, note that this is not the case for XML schemes which allow much richer structures.

In all documents the maximum depth exceeded 20 only for some specific, often heavily recursive instances. It is arguable whether it is their inherent feature or if it is a result of a lack of a good structure design. The maximum depth of corresponding schemes is higher, but it also tops around 80

	Statistics	dat	doc	ex
Doc.	Num. of elements	23,132,565	267,632	2,911,059
	Num. of attributes	33,660,779	102,945	857,691
	Num. of dist. el. names	81	134	146
	Num. of dist. rec. el. names	4	12	6
	Num. of dist. paths	434	2,086	144
	Depth of document	12	459	14
Sch.	Num. of dist. el. names	76	377	523
	Num. of dist. paths	115	11,994	1,665
	Depth of schema	12	81	79
	Statistics	**rep**	**res**	**sem**
Doc.	Num. of elements	1,957,637	21,305,818	25,548,388
	Num. of attributes	208,265	2,189,859	10,228,483
	Num. of dist. el. names	461	210	1,410
	Num. of dist. rec. el. names	0	6	1
	Num. of dist. paths	373	426	2,534
	Depth of document	6	19	11
Sch.	Num. of dist. el. names	3,213	250	-
	Num. of dist. paths	3,137	568	-
	Depth of schema	5	15	-

Table 6.4: Maximum values of global statistics

(disregarding possible recursion, i.e. an infinite depth).

Besides that, the recursion is also remarkably trivial. Mostly, the number of distinct element names with recursive occurrences is up to 3 and even the most complex documents do not have more than 12 possible recursive elements present at the same time. We will further deal with this finding in recursive statistics.

Last but not least, Table 6.5 depicts the exploitation rate of global properties, i.e. percentage of documents/schemes with at least one property. Note that the results differ substantially for categories where the XML documents are expected to be read or written by humans and for data-oriented ones. The database-oriented XML documents are designed to be more regular to ensure further simple computer processing, while human-oriented data are much more terse and also a richer syntax is used to emphasize the semantics and to improve readability. For example, it is only a matter of taste whether an attribute or simple element is used to represent the same in-

	Node type	dat	doc	ex	rep	res	sem
Doc.	Attribute	31.7	96.2	92.2	100.0	99.9	99.9
	Empty element	26.8	69.2	89.9	100.0	86.7	92.7
	Mixed element	0.2	76.5	8.7	0.0	10.1	2.4
	Recursive element	0.1	43.3	63.8	0.0	0.7	3.3
Sch.	Attribute	50.0	94.1	52.6	100.0	85.7	-
	Empty element	37.5	94.1	47.4	25.0	71.4	-
	Mixed element	37.5	100.0	50.0	0.0	57.1	-
	Recursive element	12.5	88.2	18.4	0.0	28.6	-

Table 6.5: Exploitation rate of global properties (%)

formation in the document. Similar statements hold for mixed-content or empty elements. However, as the table shows, the most common features are spread throughout all categories and thus should not be ignored by the XML processors. The only exception are mixed-content elements which are used sparsely outside the `doc` category.

Last fact to be observed is that XML features used in schemes and document instances usually match, i.e. the schemes are quite well designed in this matter, though we later show that they are too general.

Level Statistics

Level statistics focus on distribution of elements, attributes, text nodes, and mixed contents per each level of documents. Figure 6.2 depicts the distributions for the whole sample set of documents; for schemes the results are not shown, since they were too influenced by recursive and thus possibly infinitely deep documents to generate any meaningful data.

The graphs show that the highest amounts of analyzed nodes are always at first levels and then the number of occurrences rapidly decreases. The steep exponential decrease ends around level 20 and then the drop is much slower and shows more fluctuations. This correlates closely with fan-out statistics in Figure 6.3, see below.

Note that unlike attributes or mixed contents the text nodes occurrences copy the curve of element frequencies almost perfectly. This denotes that the text content is spread evenly through all levels of XML documents.

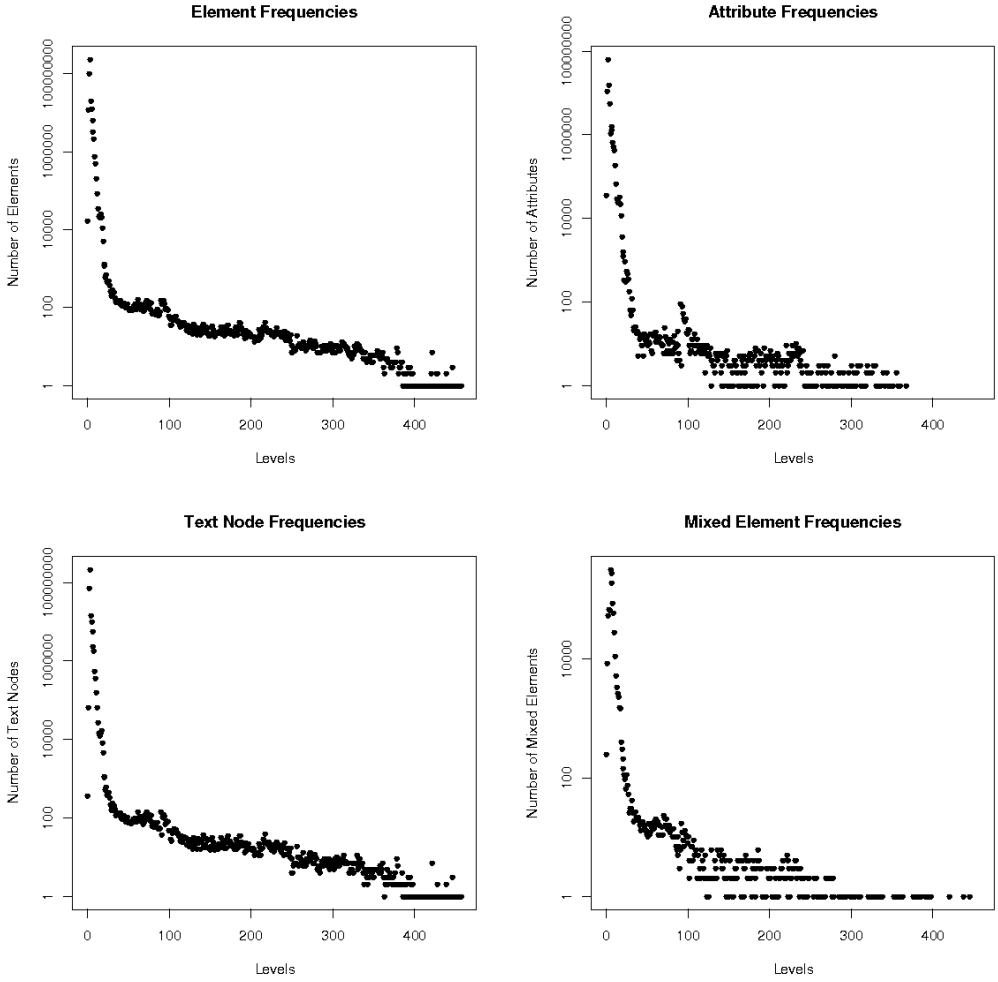


Figure 6.2: Distribution of elements, attributes, text nodes, and mixed contents in XML documents per level

Fan-Out Statistics

Fan-out statistics describe the overall distribution of XML data. Figure 6.3 depicts the results of element fan-out for XML documents per each category using 3D graphs that consist of level, fan-out value, and the number of occurrences of such pairs. Each level is for better lucidity displayed with a different color.

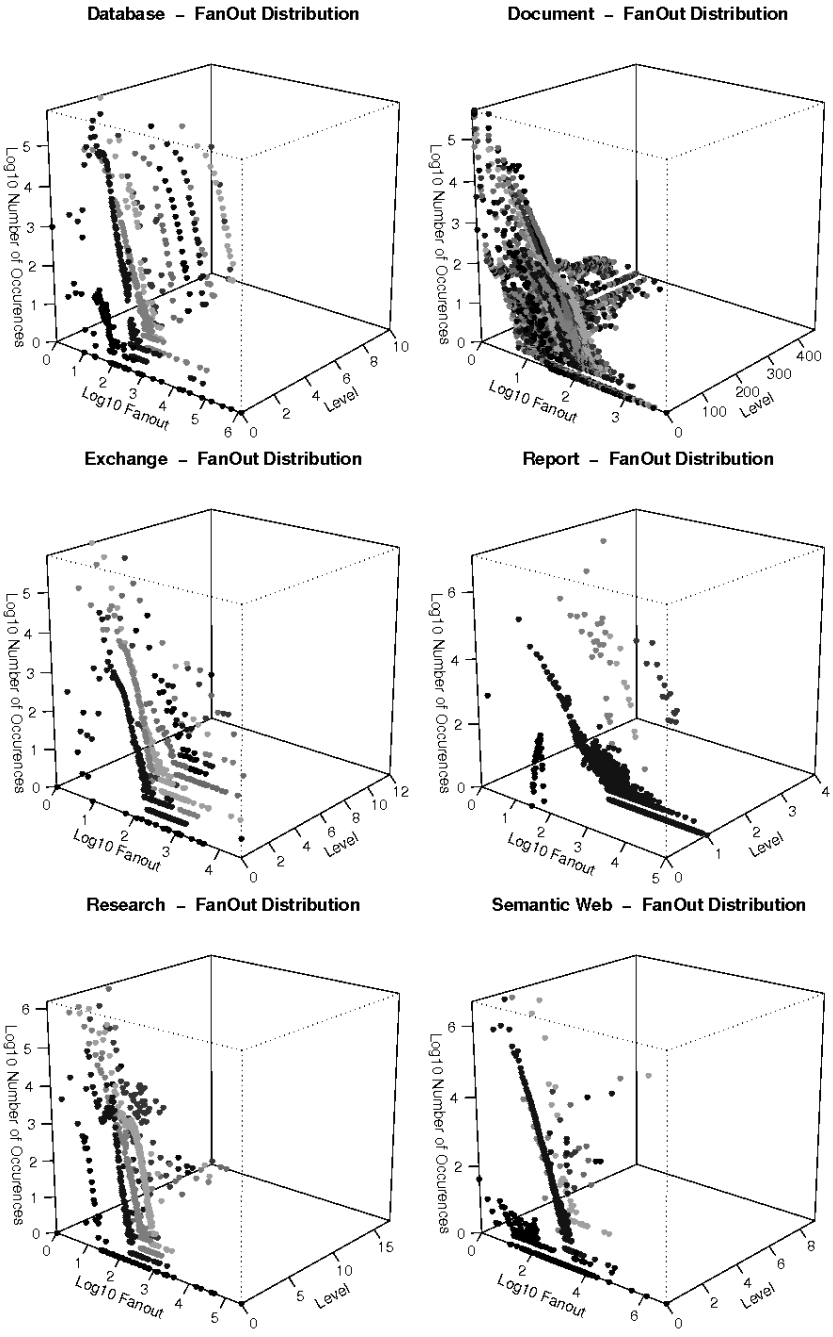


Figure 6.3: Element fan-out of XML documents per categories

We can observe that the characteristics of the graph are similar in each level, but with the growing depth it gets thinner. Similarly to level statistics the highest values are at first levels and soon radically decrease. The graphs for simple element fan-out (see [52]) are analogous, just naturally thinner. As in the previous case of element/text nodes it indicates that the distribution of trivial elements is almost same as the overall distribution of elements. It also means that simple elements are very frequent at all levels of XML documents.

Considering XML schemes the inverse fan-in values are usually rather low on average and moderate on maximum values (usually around 13 different input elements). Exceptions are the `doc` and `ex` categories which generally show far more complicated schema definitions than the rest ones (including, e.g., complete subgraphs on up to 10 nodes). For more details see [52].

Recursive Statistics

Next set of statistics, called *recursive*, deals with types and complexity of recursion. We already know that recursive elements are extensively used especially in `doc` and `ex` categories. Table 6.6 contains an overview of exploitation rates for the four previously defined types of recursion; Table 6.7 contains percentage of the respective types. Finally, Table 6.8 shows the distance of the closest and furthest ed-pairs. In all the tables **T** stands for trivial recursion, **L** stands for linear recursion, **P** stands for pure recursion, and **G** stands for general recursion.[4]

		dat	**doc**	**ex**	**rep**	**res**	**sem**
Doc.	**T**	0.06	2.38	3.67	-	0	0.27
	L	0.06	19.92	32.57	-	0.65	2.52
	P	0.03	18.76	22.48	-	0	1.46
	G	0.06	16.20	7.80	-	0.04	0
Sch.	**T**	0	0	0	-	0	-
	L	0	0	0	-	14.29	-
	P	0	2.94	7.89	-	28.57	-
	G	12.50	85.29	13.16	-	28.57	-

Table 6.6: Exploitation rate of types of recursion (%)

[4]Remember that there are no recursive elements in the `rep` category.

		dat	doc	ex	rep	res	sem
Doc.	**T**	0.2	5.0	6.4	-	0	1.0
	L	0.5	65.3	45.7	-	66.7	92.6
	P	0.7	12.7	26.9	-	0	6.4
	G	98.5	17.0	21.0	-	33.3	0
Sch.	**T**	0	0	0	-	0	-
	L	0	0	0	-	2.9	-
	P	0	0.1	1.0	-	20.6	-
	G	100.0	99.9	99.0	-	76.5	-

Table 6.7: Percentage representation of types of recursion (%)

Statistics		dat	doc	ex	rep	res	sem
Distance of closest ed-pairs	Avg.	1.9	1.5	1.6	-	2.4	1.9
	Max.	3	162	6	-	9	2
Distance of furthest ed-pairs	Avg.	1.9	3.2	2.3	-	3.6	1.9
	Max.	3	450	6	-	12	4

Table 6.8: Distance of closest and furthest ed-pairs in XML documents

It is not surprising that the recursion is mostly used in the `doc` and `ex` categories (for schema instances), while in other categories the importance of recursion seems to be only marginal (see Table 6.6).

Contrary to usual expectations according to Table 6.7 the most common type of recursion is not the general recursion but the linear recursion which consists of a single recursive element that does not branch out. The second most used type of recursion is pure recursion – still containing only one single recursive element name in the whole recursive subtree. The general recursion comprises only a lesser part of all recursion types. Finally, the trivial recursion, though occasionally present in the data, is not of any special importance. Note that these findings contradict to results of other existing papers (e.g. [29]) that claim that linear recursion is not a frequent feature and thus insignificant. This striking difference is probably caused by the semi-automatic way of gathering the data, though in the other previous cases our results were mostly similar to those in existing works.

If we compare the schema part of both the tables with the part containing results for their instances, we can see that XML schemes are probably too broad. Not only they define recursive elements when there is clearly no reason

to do so, but also they almost do not specify anything but the most general type of recursion. And neither single documents, nor whole collections do not exploit the full generality allowed by corresponding schema definitions.

The simplicity of commonly used recursion types in data instances is also apparent from Table 6.8. The average distance of the two closest recursive elements is always less than 2.5, while the average distance of the furthest pairs is between 1.9 and 3.6. The maximum values tend to be quite extreme but they only occur in very specific documents – usually in the same ones that had shown similarly peculiar features regarding the maximum depth and other similar statistics.

Mixed-Content Statistics

Mixed-content statistics further analyze the average and maximum depths of mixed content and the percentage of mixed-content and simple mixed-content elements. There is of course no point in analyzing the depth of simple mixed-content elements, since it is always equal to 1. The corresponding results are listed in Table 6.9.

Statistics		**dat**	**doc**	**ex**	**rep**	**res**	**sem**
Depth	Avg.	1.8	4.1	1.0	-	1.9	1.2
	Max.	6	448	5	-	2	3
Simple mixed contents (%)		55.9	79.4	99.6	-	1.9	78.4

Table 6.9: Mixed-content statistics for XML documents per category

Again we can observe that the structure of mixed-content elements is not complex. The average depth is low (less than 5) and most of them are even of the simplest types which consist only of trivial subelements (e.g. 55.9% for `dat`, 79.4% for `doc`, or even 99.6% for `ex` category). In this shed of light most of the currently used techniques for dealing with arbitrary mixed content seem to be unnecessarily general. It would be beneficial to handle the trivial cases separately and more efficiently.

DNA Statistics

DNA statistics focus on a new construct which we have defined on the basis of our experience with XML data. The analysis summarizes the occurrences

of such patterns and their (average and maximum) widths and depths per each category. The results are listed in Table 6.10.

	Statistics		dat	doc	ex	rep	res	sem
Doc.	Elems. involved (%)		1.19	10.66	8.08	0.00	8.64	0.61
	Num. of occurrences		91,571	296,880	179,556	3	551,806	40,017
	Width	Avg.	5.5	4.1	2.6	2.0	4.9	7.2
		Max.	57	1398	150	2	105	47
	Depth	Avg.	3.1	2.7	2.5	3.0	2.6	2.9
		Max.	9	361	8	3	17	9
Sch.	Width	Avg.	4.3	-	1.8	7.3	2.4	-
		Max.	11	-	10	26	6	-
	Depth	Avg.	3.1	-	2.3	2.0	2.4	-
		Max.	6	-	3	2	3	-

Table 6.10: DNA pattern statistics per category

The statistics show that although the pattern seems to be rather artificial, it occurs relatively often, especially in `doc` (10.66% of elements), `res` (8.64%), and `ex` (8.08%) categories. Structure of the pattern is quite simple seeing that the high maximum widths (e.g. the highest 1398) correspond to number of trivial subelements, whereas the highest maximum depths (e.g. the highest 361) seem to be rather exceptional. The average widths and depths are still quite low – lower than 10 and 3 respectively.

We believe that the pattern could be handled separately and thus more efficiently knowing that except for the exactly one degenerated branch its structure is relatively simple.

Relational Statistics

The last mentioned set of statistics focus on relational patterns – again a new construct. These patterns can easily be processed in relational databases (as simple tables) or using relational approaches, since they are often a product of various database export routines.

We analyze both relational and shallow relational patterns figuring out their number of occurrences, (average and maximum) widths, and for relational patterns also (average and maximum) fan-out of subelements, everything per each category. (In this case we do not take in results for schemes, since they are biased due to recursion.) The results are listed in Tables 6.11

and 6.12. The number of occurrences indicates how often we can match the regular relational pattern in the source data set, while the number of elements involved in the relational pattern is the total number of all elements which could be represented using a simple tabular relationship.

Statistics		**dat**	**doc**	**ex**	**rep**	**res**	**sem**
Elems. involved (%)		29.23	6.23	29.53	94.29	22.66	41.56
Num. of occurrences		170,744	154,133	185,358	40,276	619,272	716,038
Repetition	Avg.	10.5	3.3	5.8	322.7	5.1	8.8
	Max.	600,572	1,254	615	102,601	15,814	16,500
Fan-out	Avg.	3.6	1.5	2.2	6.2	2.3	3.5
	Max.	33	10	18	26	51	113

Table 6.11: Relational pattern statistics for XML documents per category

Statistics		**dat**	**doc**	**ex**	**rep**	**res**	**sem**
Elems. involved (%)		0.2	4.38	3.41	0.23	17.12	3.33
Num. of occurrences		16,025	82,255	44,403	11,957	418,342	117,834
Repetition	Avg.	4.9	6.6	5.2	44.6	14.4	14.3
	Max.	1,000	3,331	1,000	2,166	151	1,669

Table 6.12: Shallow relational pattern statistics for XML documents per category

Similarly to previous case we can see that both patterns are quite frequent (e.g. 29.23% in `dat`, 41.56% in `sem` and even 94.29% in `rep` category), though the shallow relational pattern does not have that many elements involved (almost always less than 5%). Remarkably, both the patterns are found in all categories and even though most occurrences cover only a small number of elements, there are some instances that are very large, consisting of thousands or even hundreds of thousands simple elements. Since their simple structure can easily be captured, it is probably again a good idea to handle them separately to ensure efficient processing.

6.4 Conclusion

The results of the analysis bring two important information. Firstly, they precisely describe particular characteristics and complexity of real-world XML

data from various points of view. And secondly, they bring several general conclusions. Primarily we have found out that the real data show lots of pattern usages. We have also refuted the common hypothesis that processing of recursive and mixed-content data can be omitted due to their low real occurrence. And, on the other hand, we have learned that the complexity of these two patterns is much lower than allowed by corresponding schemes. This observation holds in general – the XML schemes are too general in terms of the difference of sets of allowed and real-world instances.

Chapter 7

Query Evaluation

In this chapter we discuss problems related to query evaluation for storage strategies generated by system UserMap proposed in Chapter 4. We deal with two key issues – correction of the candidate set of annotations proposed by the GAS algorithm (see Algorithm 5) and related query evaluation. In the former case we identify and discuss situations when the proposed annotations are either meaningless or a user interaction and/or a default choice is necessary to choose from multiple possibilities. In the latter case we deal with the interface between various storage strategies and the way the system should cope with redundancy. For this purpose we have selected a sample representative set of annotations using which we illustrate the related issues and open problems. Finally, we describe and discuss the architecture of experimental implementation of the whole system.

Most of the contents of this chapter has been published in the following paper:

Mlýnková, I. – Pokorný, J.: UserMap – an Adaptive Enhancing of User-Driven XML-to-Relational Mapping Strategies. (Note: Paper under review process)

The previously described proposal focuses on deeper exploitation of information provided using schema annotations. The approach is based on the hypothesis that structurally similar schema fragments should be stored in a similar way. At this stage we have an XML schema S and a set of schema annotations F consisting of two subsets:

- F_{orig}, i.e. annotations provided originally by a user and

- F_{adapt}, i.e. annotations denoted by GAS algorithm.

The annotations from set F_{adapt} are considered as possible candidates for annotating, but not all of them should be included in the final storage strategy. Firstly, not all the candidate combinations can be applied on a schema fragment at the same time. And secondly, not all the candidate annotations have to be required by the user. As we have mentioned, a user can specify final schema fragments, i.e. fragments which should not be influenced by the GAS algorithm, but despite this feature the system can still propose candidates unappropriate for particular application. Thus the natural following step is correction of the set F_{adapt}.

Whenever the set of annotations is corrected, i.e. the final user-approved storage strategy is determined, there remains the open problem of query evaluation. Firstly, the system must cope with the interface between mapping methods, i.e. to be able to process parts of a single query using different storage strategies. This problem occurs especially in case of overriding annotation intersection (see Definition 22). But the intersection of schema fragments can be also redundant or influencing (see Definitions 21 and 23) meaning that a single schema fragment can be stored using two (or more) strategies at the same time. Therefore the system must be also able to efficiently determine which of the available storage strategies should be used for evaluation of a particular query.

7.1 Related Work

Firstly, let us consider the existing works from the point of view of checking correctness of the resulting mapping strategy and query evaluation.

Papers [34] [22] which introduce system ShreX also propose definitions of a *correct* and *lossless* mapping. In the former case it means that the mapping produces a valid relational schema in terms of distinct table names, distinct column names within a table, distinct CLOB names, and existence of at least one key in each table. In the latter case lossless mapping is a mapping which is correct and maps each element and attribute of the schema and the sibling order of elements. The system is able to check the correctness and losslessness of the annotations and complete incompleteness using default mapping rules. As for the query evaluation the system ShreX does not support redundancy and thus the choice of the most efficient storage strategy for a particular

query is pointless. The interface between storage strategies is solved using a mapping API and a mapping repository. The repository is used for storing information about how each element and attribute is stored, which mapping is used to capture the document structure, and which tables are available in the relational schema. Hence the system is able to get information about storage strategy for any part of the schema and thus shred the documents as well as evaluate queries.

On the other hand, the system XCacheDB [23] supports only a single strategy for shredding XML data into tables which can be modified by inlining / outlining of a fragment, storing a fragment to a BLOB column, and specification of corresponding table and column names and data types. Hence, the only incorrect combination is concurrent inlining and outlining of the same schema fragment that can be detected easily. The information about the structure of the current schema is again stored into the database. Contrary to ShreX, the XCacheDB system allows a kind of redundant annotation intersection enabling to store a schema fragment to a BLOB column and, at the same time, to shred it into a set of tables which needs to be treated in a special way. The proposed enhancing of a classical query evaluator is quite simple but working. It always chooses the query plan with minimal number of joins.

As it is obvious, in both the cases the set of possible situations is somehow simplified. In the former case the set of annotation intersections is restricted, whereas in the latter case the set of mapping strategies can be characterized as a set of modifications of a single mapping strategy. But in our case we consider all the three types of intersections and more complex combinations of mapping strategies.

7.2 Correction of Candidate Set

The first step related to efficient query evaluation is correction of the candidate set of annotations F_{adapt}. Apart from checking correctness and completeness [34] of schema annotations, we can distinguish three cases corresponding to the following three steps:

1. Since not all the proposed annotations can be used at the same time, the system removes cases which are forbidden or meaningless.

2. The system identifies cases where the user can choose from several possibilities if the default choice is not satisfactory.

3. The system accepts further user-specified corrections of proposed annotations which do not correspond to intended future usage.

In the first two cases the system must be able to correctly identify the situations, the last case is rather the question of user-friendly interface.

7.2.1 Missed Annotation Candidates

Despite the above described cases, we can also identify situations when the system could automatically add more annotation candidates. We discuss them in the following examples, where f_A, f'_A, f_B, f'_B denote schema fragments f and f' annotated using storage strategies A or B respectively.

Situation A – Unidentified Annotated Subfragment

Let us consider the situation depicted in Figure 7.1 where schema S is provided with a set of annotated fragments $F_{orig} = \{f_A, f_B\}$, where $f_B \subset f_A$, whereas the GAS algorithm identified a set of fragments $F_{adapt} = \{f'_A\}$ (as depicted by schema S'). The question is whether it is necessary to add also fragment f'_B, where $f'_B \subset f'_A$ (as depicted by schema S'').

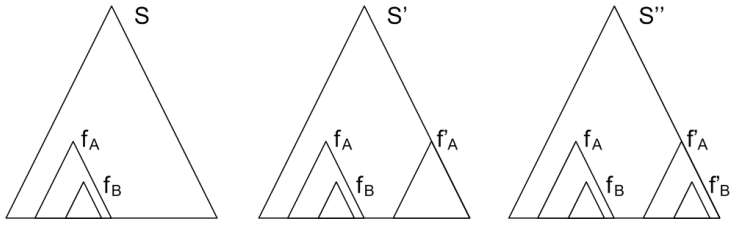


Figure 7.1: Unidentified annotated subfragment

If we analyze the situation, the answer is obvious. If $sim(f_B, f'_B) > T_{sim}$, the algorithm would add f'_B to F_{adapt} too. Thus if f'_B is not annotated, its similarity is not high enough and thus it should not be added to F_{adapt}.

Situation B – Unidentified Annotated Superfragment

The second situation is depicted in Figure 7.2. Schema S is again provided with a set of annotated fragments $F_{orig} = \{f_A, f_B\}$, $f_B \subset f_A$, but the GAS algorithm identified a set of fragments $F_{adapt} = \{f'_B\}$ (as depicted by schema S'). In this case we do not discuss whether it is necessary to add also fragment f'_A, where $f'_B \subset f'_A$, because we can apply the same reason as in case of Situation A. The question is whether it is necessary to add also fragment f'_A, where $f'_B = f'_A$ (as depicted by schema S'').

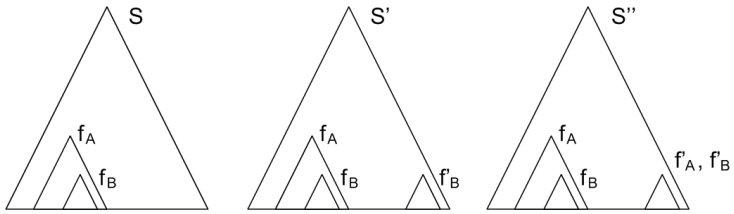


Figure 7.2: Unidentified annotated superfragment

In this case we cannot simply state which of the possibilities should be chosen, because for both we can find good reasons. Thus this is the case for user interaction and/or a default setting.

Note that in case $f_A = f_B$ then, obviously, $f'_A = f'_B$, i.e. the system should apply both the annotations.

7.2.2 Sample Set of Annotations

For demonstration of further open issues related to correction of candidate set of annotations as well as query evaluation, we have chosen particular types of fixed mapping methods. They represent typical and verified storage strategies and, at the same time, enable to exploit all the three types of annotation intersections. The whole set of supported annotating attributes, their values, and corresponding mapping strategies is listed in Table 7.1.

Similarly to the existing works (see Chapter 3.2) we support inlining and outlining of a schema fragment to/from parent table or its storing to a single CLOB column. As for the "classical" mapping methods we support a set of schema-oblivious storage strategies – the Edge, Attribute, and Universal mapping [35] – and a set of schema-driven storage strategies – the Basic, Shared, and Hybrid algorithm [67]. Last but not least, we support a kind

Attribute	Value	Function
INOUT	inline, outline	Specifies whether the annotated fragment should be inlined or outlined to/from parent table.
GENERIC	edge, attribute, universal	The annotated fragment is stored using the specified type of generic-tree mapping strategy [35], i.e. Edge, Attribute, or Universal mapping.
SCHEMA	basic, shared, hybrid	The annotated fragment is stored using the specified type of schema-driven mapping strategy [67], i.e. Basic, Shared, or Hybrid mapping.
TOCLOB	true	The annotated fragment is stored to a CLOB column.
INTERVAL	true	The annotated fragment is indexed using the Interval encoding [73].

Table 7.1: Supported schema annotations

of numbering schema which speeds up processing of particular queries – the Interval encoding [73].

Contrary to existing works we do not consider annotations which enable to specify names of tables and columns or data types of columns (except for the CLOB one), since these are related rather to user-friendliness of the system and do not require special or complicated treatment (except for checking correctness).

Naturally, the set of supported mapping strategies could be much wider and involve more representatives of the existing reasonable mapping strategies. But our aim was to choose well-known representatives of particular approaches which enable to illustrate various situations.

7.2.3 Annotation Intersection

In Section 4.1.1 we have defined three types annotation intersection – overriding, redundant, and influencing – assuming that the system is provided with both the set of annotations and types of their mutual intersection. But there are also cases when a particular combination of annotations is senseless. For instance consider the situation depicted in Figure 7.3, where schema S contains two annotated fragments f_{TOCLOB} and f_{SCHEMA}, whereas $f_{SCHEMA} \subset f_{TOCLOB}$ and the TOCLOB annotation overrides all the previously

specified strategies. As it is obvious, such combination of annotations is useless, since there is no point in shredding a part of a schema fragment stored in a CLOB column into a set of tables. The situation also depicts that the order of composition of annotations is important. Obviously, the opposite order, i.e. if $f_{TOCLOB} \subset f_{SCHEMA}$, is reasonable and can result in both redundant and overriding intersection.

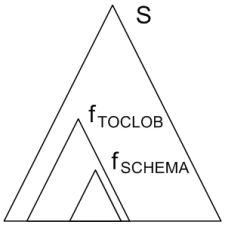


Figure 7.3: Forbidden intersection of annotations

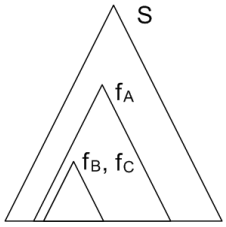


Figure 7.4: Intersection of multiple annotations I.

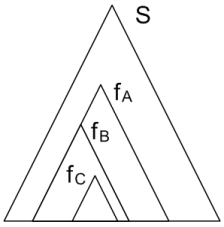


Figure 7.5: Intersection of multiple annotations II.

Another question is for which subsets of the supported schema annotations should the intersection type be specified. Consider the situation depicted in Figure 7.4, where schema S contains three annotated fragments f_A, f_B, and f_C, whereas $f_B = f_C$ and $f_B, f_C \subseteq f_A$. At this situation we naturally need to know the result of intersection of all the three annotations together. And with the above finding, it can also differ depending on the order of their mutual composition. Therefore, we should theoretically specify the result of intersection of all possible subsets of the annotation set Ω_A and all respective orders. But, in fact, as there are pairs of annotations or their orders which are forbidden, the number of such specifications significantly decreases. And, in addition, such specifications need to be stated for the whole system only once, not for each mapping task separately.

We demonstrate both the situations for the sample set of annotations in the following chapters.

Intersections of Pairs of Annotations

The specification of allowed types of intersections for pairs of annotations is relatively simple. Four our sample set of annotations they are listed in Tables 7.2 and 7.3, where $\emptyset$ represents no effect of intersection, $\times$ represents forbidden intersection, and ✓represents allowed intersection. Each field of

the table represents the result of applying the mapping strategy in the row on the mapping strategy in the column.

	INOUT	GENERIC	SCHEMA	TOCLOB	INTERVAL
INOUT	∅	×	×	×	×
GENERIC	×	∅	✓	×	×
SCHEMA	×	✓	∅	×	×
TOCLOB	×	✓	✓	∅	×
INTERVAL	×	×	×	×	∅

Table 7.2: Overriding and redundant annotation intersection

	INOUT	GENERIC	SCHEMA	TOCLOB	INTERVAL
INOUT	∅	✓	✓	×	×
GENERIC	×	∅	×	×	×
SCHEMA	×	×	∅	×	×
TOCLOB	×	×	×	∅	×
INTERVAL	×	✓	✓	×	∅

Table 7.3: Influencing annotation intersection

As can be seen from the tables, the amount of reasonable and thus allowed combinations of mapping methods is relatively low in comparison with the theoretically possible options. Firstly, we assume that the combination of two identical annotations results in an empty operation, i.e. it has no effect as depicted at diagonals. And, in addition, in our case the results for overriding and redundant annotations are identical and thus listed in one common table.

Another two obvious cases are `INOUT` and `INTERVAL` annotations which are supposed to influence any other method. Therefore, they can occur only in case of influencing intersections and only in one particular order of composition. Naturally, they can be applied only on methods which shred a schema fragment into a set of tables.

Considering the `TOCLOB` annotation, it can be also applied on a method which shreds a schema fragment into one or more tables, since the whole fragment is then viewed as a single attribute. As for the composition orders, as depicted in Figure 7.3, the `TOCLOB` annotation can be applied only on a mapping strategy, but not vice versa. Note that from another point of view the `TOCLOB` annotation can be regarded as influencing, rather than overriding. Similarly to the `INOUT` annotation it influences the given storage strategy treating a schema fragment as a single attribute. But it is only a question of semantics.

Last but not least, note that if the intersecting fragments are equivalent, i.e. the order of composition of mapping strategies is not obvious, we take as the result union of both the possible orders.

Intersections of Multiple Annotations

As for the intersection of multiple annotations together we need to distinguish several cases. We demonstrate them using the example depicted in Figure 7.5, where schema S contains three annotated fragments f_A, f_B, and f_C, whereas $f_C \subset f_B \subset f_A$. The question is what will be the result of annotation for their intersection.

As for the first situation let us assume that the intersection of f_A and f_B is overriding. Than the situation transforms to the case of intersection of two methods (i.e. f_B and f_C) as defined in the previous chapter.

The second situation occurs when the intersection of f_A and f_B is redundant, meaning that the common schema fragment is stored using both the strategies A and B. Then the situation of intersection with strategy C transforms to union of separate intersections of two pairs of methods (i.e. f_A, f_C and f_B, f_C). Or, also in this case the user can specify on which of the two strategies A and B should strategy C be applied.

The third situation involves the last case when the intersection of f_A and f_B is influencing. In this case the resulting intersection must be defined for all the possible cases. For our sample set of annotations, the solution again corresponds to union of separate intersections of two pairs of methods (whereas one of them always results in forbidden intersection). But, in general, the result can lead to a brand new one method and therefore a new set of rules for intersecting.

Note that also in this case if the intersecting fragments are equivalent, i.e. the order of composition of mapping strategies is not obvious, we again take as the result union of all the possible orders. And similar discussion can be done for larger sets of annotations as well.

Multiple Annotation Candidates

As it is obvious, there are situations when multiple options for determining the resulting storage strategy are available. At first glance a solution would be a set of priorities assigned to the possibilities. But the problem is that such specification cannot be done beforehand, since priorities suitable for one type of schema fragment and particular application do not have to suit another one. Therefore a user intervention and/or a default choice should be exploited also in these cases.

7.2.4 Examples of Schema Annotations

The annotations supported by system UserMap (see Table 7.1) are expressed using attributes from a name space called *usermap* and can be associated with element definitions of XSDs. As we have mentioned, similarly to paper [34] they could be associated also with attributes, attribute groups, element groups, etc. But we restrain to elements for simplicity.

Example 1 – Exploitation of CLOBs

The exploitation of CLOBs enables to speed up reconstruction of schema fragments. It is useful especially in cases when the user knows that particular schema fragment is rather document-oriented and will be retrieved as a whole. A similar idea is exploited in paper [39] which automatically shreds structured parts of a schema into relations and stores semi-structured ones into a single text column with the support of XML-aware full-text operations.

Consider the example in Figure 7.6 where a fragment of XSD of the *Internet Movie Database (IMDb)*[1] contains information about actors. Each actor has name consisting of first name and last name and filmography, i.e. a list of movies each consisting of title and year. For better lucidity the element names are underlined and schema annotations are in boldface.

```
<xs:element name="Actor"
            usermap:SCHEMA="hybrid">
 <xs:complexType>
  <xs:sequence>

   <xs:element name="Name"
               usermap:TOCLOB="true">
    <xs:complexType>
     <xs:sequence>
      <xs:element name="FirstName" type="xs:string"/>
      <xs:element name="LastName" type="xs:string"/>
     </xs:sequence>
    </xs:complexType>
   </xs:element>

   ...

   <xs:element name="Filmography">
    <xs:complexType>
     <xs:sequence>
      <xs:element name="Movie" maxOccurs="unbounded">
       <xs:complexType>
        <xs:sequence>
         <xs:element name="Title" type="xs:string"
                     usermap:INOUT="outline"/>
         <xs:element name="Year" type="xs:int"/>
        </xs:sequence>
       </xs:complexType>
      </xs:element>
     </xs:sequence>
    </xs:complexType>
   </xs:element>

  </xs:sequence>
 </xs:complexType>
</xs:element>
```

Figure 7.6: Exploitation of CLOBs – XML schema

[1] `http://www.imdb.com/`

According to the specified annotations, the whole fragment, i.e. element `Actor`, should be stored using the Hybrid algorithm, as specified by the `SCHEMA="hybrid"` annotation, its subelement `Name` should be stored into a CLOB column (`TOCLOB="true"`), and subelement `Title` should be outlined to a separate table (`INOUT="outline"`).

As depicted in Tables 7.2 and 7.3 the intersection of `SCHEMA` and `TOCLOB` annotations can be either overriding or redundant. Firstly, let us consider the overriding case. The resulting relational schema is depicted in Figure 7.7 (b).

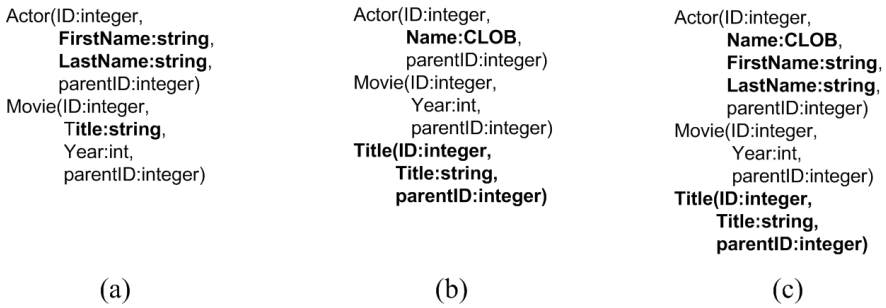


Figure 7.7: Exploitation of CLOBs – relational schemes

As it is expectable the resulting relational schema corresponds to the result of classical Hybrid algorithm (depicted in Figure 7.7 (a)) except for two cases. Firstly, the element `Name` is treated as an element having a text content and stored into a single CLOB column. And secondly, the element `Title` is stored into a separate table, although the classical Hybrid algorithm would inline it to table `Movie` too.

Example 2 – Redundant Mapping Strategies

Let us again consider the XSD example in Figure 7.6, but this time assuming that the intersection of `SCHEMA` and `TOCLOB` annotations is redundant. Though the redundancy in general leads to a significant space overhead, it can have reasonable applications, where the need of retrieval efficiency exceeds this disadvantage. The resulting relational schema is depicted in Figure 7.7 (c). In this case the element `Name` is stored twice, using both the strategies, i.e. into two columns corresponding to classical Hybrid algorithm and, at the same time, into one CLOB column.

Note that a similar type of storage strategy can be defined also using the system XCacheDB [23] which enables both redundant and overriding mapping to a CLOB column. The main difference is that the system supports only one particular type of shredding into a set of tables which can be modified by inlining and outlining.

Example 3 – Influencing Mapping Strategies

Last but not least, consider an example of influencing intersection of mapping strategies. In example depicted in Figure 7.8 we use the same fragment of IMDb XSD, but with different annotations. This time the element `Actor` should be stored using schema-oblivious Edge mapping (`GENERIC="edge"`), whereas queries over its subelement `Filmography` are enhanced using the Interval encoding (`INTERVAL="true"`).

```
<xs:element name="Actor"
            usermap:GENERIC="edge">
 <xs:complexType>
  <xs:sequence>

   <xs:element name="Name">
    <xs:complexType>
     <xs:sequence>
      <xs:element name="FirstName" type="xs:string"/>
      <xs:element name="LastName" type="xs:string"/>
     </xs:sequence>
    </xs:complexType>
   </xs:element>

   ...
```

```
<xs:element name="Filmography"
            usermap:INTERVAL="true">
 <xs:complexType>
  <xs:sequence>
   <xs:element name="Movie" maxOccurs="unbounded">
    <xs:complexType>
     <xs:sequence>
      <xs:element name="Title" type="xs:string"/>
      <xs:element name="Year" type="xs:int"/>
     </xs:sequence>
    </xs:complexType>
   </xs:element>
  </xs:sequence>
 </xs:complexType>
</xs:element>

 </xs:sequence>
</xs:complexType>
</xs:element>
```

Figure 7.8: Influencing mapping strategies – XML schema

Figure 7.9 depicts both the classical Edge mapping (a) and the result of the strategy specified by the annotations (b). In the former case all the edges of the graph are stored into a single table `Edge`. In the latter case edges of subelement `Filmography` are stored into a separate table `EdgeFilmography` having additional columns (`intervalStart` and `intervalEnd`) for storing values of Interval encoding. Note that the influencing enables to skip the column `order`, since the Interval encoding involves total ordering.

The closest example can be found in case of system ShreX [34] which supports annotation `structurescheme` specifying how the structure of the

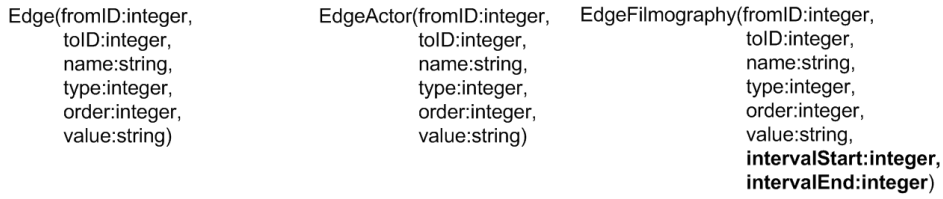


Figure 7.9: Influencing mapping strategies – relational schemes

whole schema is captured, i.e. using keys and foreign keys, Interval encoding, or Dewey decimal classification. This feature can be considered as a special type of influencing mapping, though its purpose is slightly different.

Auxiliary Columns

As the above described examples of relational schemes are illustrative, they only represent the characteristics of the mapping strategies. In fact, when the schemes are generated automatically, we cannot use directly, e.g., the element and attribute names for table and column names of the schema. In addition, each of the data tables contains also auxiliary information. The most important ones are a unique ID of the document the data originate from (`docID`) and a unique ID of each record (`recordID`). The former one enables to distinguish where the data originate from, but it is in the following examples omitted for simplicity. The latter one is exploited in two cases for similar reason. Since each of the storage strategies can have structurally different tables we use this uniform ID to enable their joining, as well as for uniform resulting value of the SQL query. We illustrate its usage in the following text.

7.3 Query Evaluation

The basic idea of XML query evaluation in (O)RDBMS-based storage strategies is relatively simple. An XML query posed over the data stored in the database is *translated* to a set of SQL queries (which is usually a singleton) and the resulting set of tuples is transformed to an XML document. We speak about *reconstruction* of XML fragments. As it is analyzed in detail in [42], the amount of works which focus on efficient XML-to-SQL query trans-

lation is enormous (the analysis considers about 40 papers), can be classified according to various purposes, and focus on various aspects of the problem. Two key metrics for query evaluation are *functionality*, i.e. the variety of types of supported XML queries, and *performance*, i.e. the efficiency of evaluation of the query.

The main idea of our proposed system is to enable a user to create a hybrid XML-to-relational storage strategy, i.e. a relational schema which consists of multiple subschemes having different structure. Assuming that each of the subschemes can (and usually does) require a different XML-to-SQL query translation algorithm, we focus on the problem of interface between the storage strategies, i.e. the problem of evaluation of parts of a single XML query using various storage strategies. Second issue is related to redundancy that can occur in case of intersection of annotations, in particular redundant and influencing ones, where a single schema fragment is stored using two or more strategies. Therefore a natural assumption is that the system can estimate the cost of query evaluation using all possible strategies and choose the optimal one. For this purpose we again exploit our sample set of annotations (see Table 7.1) and using simple examples we illustrate the related issues.

7.3.1 Interface between Schema Annotations

Let us consider the three types of annotation intersections separately and discuss their difference from the point of view of query evaluation. In case of overriding intersection of strategies A and B, the interface must allow joining (one or more) tables of strategy A with (one or more) tables of strategy B. In case of redundant intersection of strategies A and B the situation is similar but, in addition, the interface must enable to use any of the strategies. The last type of annotation intersection, i.e. the influencing one, requires a brief discussion: Under a closer investigation we can see that there are two types of annotations, i.e. mapping strategies, that can influence another one. Each of them is represented by one of the annotations of the sample set. The `INOUT` annotation enables to modify the structure of the resulting storage strategy, i.e. the amount of tables and/or columns. Therefore it is processed before the schema is mapped to relations and having the information about the structure it does not need to be taken into account later. We call these annotations *early binding*. (Note that the `TOBLOB` annotation can be viewed as a kind of early binding annotation as well.) On the other hand, the `INTERVAL`

annotation enhances a given storage strategy with additional information which is exploited as late as a query is evaluated. We call these annotations *late binding*.

Structural Tables

Since the resulting storage strategy is not fixed and can be influenced by many factors, we need to store the information about the structure of each mapped XML schema. Similarly to papers [34] [23] we store the information into supplemental tables. This simple idea enables to parse the schema annotations only once and not every time a document is shredded into relations or a query is posed, as well as it enables to make the processing to be independent on the way the mapping is specified.

In particular for the sample set of annotation strategies we use the following tables:

`xmlSchemaTable(uri,schemaID)`

contains information about XML schemes for which there exists a mapping in the repository, i.e. URI of the XML schema (`uri`) and ID of the schema (`schemaID`) unique within all the stored schemes.

`xmlDocTable(schemaID,url,docID)`

that contains information about XML documents stored in a database schema created for particular XML schemes, i.e. URL of the XML document (`url`) and ID of the document (`docID`) unique within all the documents valid against a schema (`schemaID`) stored in the repository.

`xmlAttrTable(schemaID,attrID,mapID,xmlName,paramID)`

contains information about storage strategies for particular attributes in the schema, i.e. ID of the attribute (`attrID`) unique within a schema (`schemaID`), ID of the storage strategy for the attribute (`mapID`), name of the attribute (`xmlName`), and ID of additional parameters (`paramID`) corresponding to the respective storage strategy, or null if there is no such information needed.

Note that the attributes could be treated similarly to elements with text content, i.e. different storage strategies could be defined for an element and for its attributes. But for simplicity the experimental implementation assumes that the storage strategy for attributes is always denoted by the storage strategy for corresponding element.

`xmlElemTable(schemaID,elemID,mapID,xmlName,paramID)`

contains information about storage strategies for particular elements in the schema, i.e. ID of the element (`elemID`) unique within a schema (`schemaID`), ID of the storage strategy for the element (`mapID`), name of the element (`xmlName`), and ID of additional parameters (`paramID`) corresponding to the respective storage strategy, or null if there is no such information needed.

`xmlElemAttrTable(schemaID,elemID,attrID)`

contains information about relationship between an element (`elemID`) and its attribute (`attrID`) in a schema (`schemaID`).

Note that this information could be stored into the `xmlAttrTable` as well, but assuming that XSDs enable to specify global attributes and attribute groups which can be referenced from several elements, the relationship can be in general M:N.

`xmlElemElemTable(schemaID,elemID,subElemID)`

contains information about M:N relationship between an element (`elemID`) and its subelements (`subElemID`) in a schema (`schemaID`).

Last but not least, there remains the structure of tables containing the additional parameters necessary for particular mapping strategies. Since each of the strategies can require different information we have outlined the parameters from tables describing elements and attributes and we store it separately, though the relationship between the tables is 1:1.

If we consider the sample set of annotations, we obviously do not need any other information for the `TOCLOB` and `INOUT` annotations themselves, since both of them are early binding annotations which influence the amount of tables of another strategy.

As for the `GENERIC` annotation, in case of pure Edge and Universal mapping the target schema is fixed regardless the source data. But in case of Attribute mapping which requires a separate table for each distinct element or attribute name in the schema, or when inlining or outlining is applied on any of the three cases, we need to know the name of table where the element / attribute is stored.

In case of `SCHEMA` annotation the situation is particularly complicated since the resulting relational schema is given by the structure of the source XML schema. And, in addition, `TOCLOB` or `INOUT` annotations can be applied

on any of the three algorithms and influence the structure as well. Therefore, for each element and attribute we need to store the information where and how it is stored. For this purpose we use table

```
xmlBSHTable(mapID,mapType,tableName,columnName)
```

that contains mapping type (`mapType`) of the element / attribute, where `n` denotes numeric data type, `s` denotes string data type, and `e` denotes element content (of an element), name of table for storing the element / attribute (`tableName`), and name of column for storing the element / attribute (`columnName`).

Last but not least, there remains the late binding `INTERVAL` annotation. As we have mentioned it is considered as an additional index that can speed up processing of particular approaches. Hence for element and attributes enhanced with this index we need to know the names of columns where the corresponding values are stored.

An example of content of structural tables for the three relational schemes (a), (b), (c) in Figure 7.7 is depicted in Figure 7.10. Since there are no attributes in the sample schema, we only deal with structural tables related to elements. The processing of attributes would be very similar. (We do not use particular IDs of the 1:1 relationship for simplicity; the related tables are mentioned in separate rows.)

As it is obvious, the tables contain all the information necessary for both document shredding and query evaluation. For each element of the sample schema we know its storage strategy and related details, i.e. the way it is stored and the target table and/or column. Note the way we treat redundant intersection of annotations. In this case we create two instances of the redundant fragment, each having its own ID and type of storage strategy.

7.3.2 Document Shredding

Having the information from structural tables, the process of document shredding is relatively simple. For instance, if we consider the relational schemes in Figure 7.7, for storing an element we generally need two information – ID of its parent element and constructors of its subelements that are mapped to the same table. Therefore the process can be described as a recursive creation of constructor for the current element from constructors of its subelements (and attributes). During the top-down progress the ID

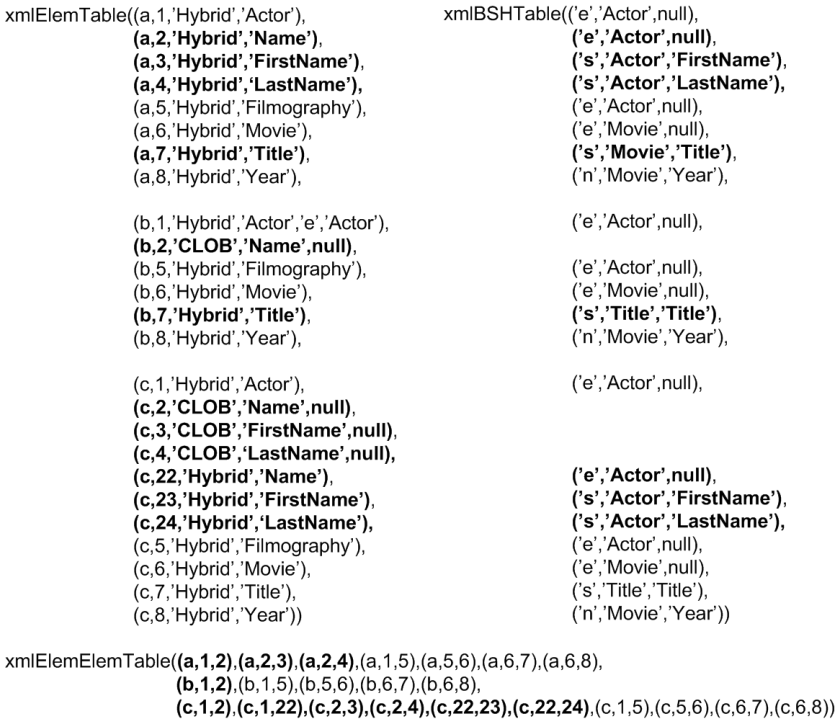


Figure 7.10: Exploitation of CLOBs – structural tables

of the current element e is propagated to be stored in `parentID` columns of its subelements $e_1, e_2, ..., e_k$ mapped to tables. During the bottom-up return from the recursion its subelements $e_{k+1}, e_{k+2}, ..., e_n$ mapped to simple types hand over their constructors and column names. Then element e creates its own constructor.

For instance, consider the element `Movie` in relational schema (b) in Figure 7.7. Being provided with parent ID, the shredding process first identifies that the element `Movie` has element content. Therefore it generates its ID and recursively processes its subelements. The subelement `Title` stores itself to own table, but the subelement `Year` returns the constructor of the integer value and name of the corresponding column. Hence the constructor of the `Movie` element consists of three items – `ID`, `Year`, and `parentID`.

7.3.3 Query Evaluation

Similarly to the process of document shredding, with the information from structural tables the query evaluation is quite straightforward. Using the following examples we illustrate the related key ideas. Assuming that wild-card queries are usually converted into union of several simple path queries with predicates, one for each satisfying wild-card substitution [41], we consider only examples of simple-path queries.

Example 1 – Early Binding Annotations

Let us consider the example of query

```
/Actor/Filmography/Movie[Year=2007]/Title
```

and relational schema depicted in Figure 7.7 (b). Firstly, the table where element `Actor` is stored, is added to the `FROM` clause. Seeing that element `Filmography` is mapped to the same table, this step has no effect. Then, since the element `Movie` is mapped to its own table, the table `Movie` is added to the `WHERE` clause and joined using IDs. The processing of predicate `Year=2007` first requires analysis of the query on the left. Since element `Year` is mapped to column of the `Movie` table, it does not cause new joins, but only adding the condition to the `WHERE` clause. The `Title` element is again detected to be stored in a separate table resulting in another join. Finally, the `SELECT` clause is provided with the reference to its record ID, resulting in the following query:

```
SELECT t.recordID
FROM Actor a, Movie m, Title t
WHERE m.parentID = a.ID AND
      m.Year = 2000 AND
      t.parentID = m.ID
```

Note that using the knowledge of semantics of the XML schema, i.e. on the basis of analysis of structural tables, this query could be further optimized, in particular, table `Actor` can be omitted.

From the point of view of evaluation of a single query using several storage strategies, the described example of query evaluation copes with an early binding influencing annotation `INOUT` applied on annotation `SCHEMA`. Thus

the translation approach is similar to classical Hybrid algorithm, except for slightly different structure which is captured using the structural tables. A similar effect would occur in case of overriding annotation `TOCLOB`.

Example 2 – Structurally Different Tables

Now, let us consider the same query but a situation, where element `Actor` is associated with annotation `SCHEMA="hybrid"` and element `Movie` with annotation `GENERIC="edge"` as depicted in Figure 7.11, whereas the intersection is overriding.

```
<xs:element name="Actor"
            usermap:SCHEMA="hybrid">
 <xs:complexType>
  <xs:sequence>

   <xs:element name="Name">
    <xs:complexType>
     <xs:sequence>
      <xs:element name="FirstName" type="xs:string"/>
      <xs:element name="LastName" type="xs:string"/>
     </xs:sequence>
    </xs:complexType>
   </xs:element>

   ...
```

```
<xs:element name="Filmography">
 <xs:complexType>
  <xs:sequence>
   <xs:element name="Movie" maxOccurs="unbounded"
               usermap:GENERIC="edge">
    <xs:complexType>
     <xs:sequence>
      <xs:element name="Title" type="xs:string"/>
      <xs:element name="Year" type="xs:int"/>
     </xs:sequence>
    </xs:complexType>
   </xs:element>
  </xs:sequence>
 </xs:complexType>
</xs:element>

 </xs:sequence>
</xs:complexType>
</xs:element>
```

Figure 7.11: Join of structurally different tables – XML schema

Hence, we need to join structurally different tables whose IDs have quite different meaning. As for the first part of the query `/Actor/Filmography` the translation remains the same as in the previous example. As for the second part of the query `/Movie[Year=2007]/Title`, we need to join the `Edge` table three times, i.e. for `Movie` and `Title` elements and for the predicate `Year=2007` resulting in a query:

```
SELECT t.recordID
FROM Edge m, Edge y, Edge t
WHERE m.toID = y.fromId AND
      y.value = 2000 AND
      m.toID = t.fromId
```

Finally, for the purpose of joining the two tables, i.e. `Actor` and `Edge`, we exploit the auxiliary `recordID` column of both the tables and information from another structural table

```
xmlInterTable(docID,anotID,parentID)
```

which contains pairs of parent-child relationships between `recordID` of an annotated element (`anotID`) and `recordID` of its parent element (`parentID`). Then the resulting query translation is as follows:

```
SELECT t.recordID
FROM Actor a, xmlInterTable i, Edge m, Edge y, Edge t
WHERE a.recordID = i.parentID AND m.recordID = i.anotID AND
      m.toID = y.fromId AND
      y.value = 2000 AND
      m.toID = t.fromId
```

The example depicts that a join with `xmlInterTable` is added every time the query "passes borders" of two mapping strategies. The obvious exception is the case of early binding influencing annotations.

Naturally more complex mapping strategies could require another information about their mutual interface, but for our particular sample set this information is sufficient.

7.3.4 Exploitation of Redundancy

The above described algorithm of query evaluation assumes that there is always one possible way it can be performed. Naturally each SQL query can have multiple query plans, each having its cost depending on the order tables are joined, selectivity of `WHERE` conditions, usage of `ORDER BY` clauses, etc. But in this chapter we deal with the set of distinct mapping strategies that "cover" the query.

Consider again the same sample query and the annotated schema in Figure 7.11, where element `Actor` is associated with annotation `SCHEMA="hybrid"` and element `Movie` with annotation `GENERIC="edge"`, whereas the intersection is now redundant. Then we have two possibilities how to evaluate the query – either using purely the Hybrid mapping or using both the Hybrid and Edge mapping and their interface. In the former case the query would

require joining of two tables – `Actor` and `Movie` (the classical Hybrid algorithm inlines the element `Title`). The query translation of the later case was discussed above and involves joining of five tables – table `Actor` of the Hybrid mapping, three `Edge` tables from the Edge mapping, and table `xmlInterTable` carrying the interface information between the two strategies. If we use a simple cost metric which considers purely the amount of join operations necessary for query evaluation, the former translation strategy is naturally better choice.

In general, there can exist a plenty of possibilities how to evaluate a query Q. For this purpose we first analyze the structural tables and search for all the possible sequences of strategies using which Q can be evaluated and we build and auxiliary *evaluation graph* G^{eval}. Consider the sample situation in Figure 7.12.

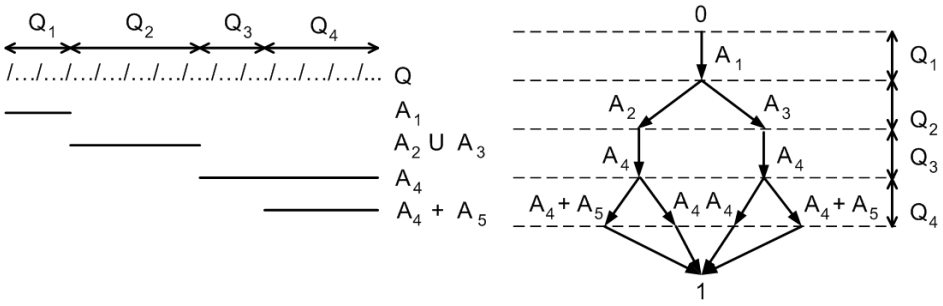


Figure 7.12: Example of evaluation graph G^{eval}

The figure schematically depicts that query Q is divided into fourth parts Q_1, Q_2, Q_3, and Q_4, determined by four annotations, i.e. mapping strategies it "traverses". Part Q_1 can be evaluated only using strategy A_1. Part Q_2 can be evaluated either using strategy A_2 or A_3 meaning that the intersection of the two annotations is redundant (denoted by the union sign), but they override annotation A_1. As for the part Q_3 the respective strategy is again only A_4 which overrides the previous two strategies A_2 and A_3. And finally, part Q_4 can be evaluated using both A_4 or influencing intersection of A_4 and A_5 (denoted by the plus sign).

On the right-hand side of the figure is depicted corresponding evaluation graph G^{eval} whose edges correspond to storage strategies and nodes to interfaces among them. The graph also contains two auxiliary nodes 0 and 1 which represent the beginning and end of the query and respective auxiliary

edges.

The construction of G^{eval} is relatively simple:

1. The auxiliary node 0 is created.
2. Starting with the set of storage strategies $A = \{A_1, A_2, ..., A_k\}$ for the root element of query Q, respective k outgoing edges of node 0 are created.
3. Each edge and corresponding storage strategy A_i is processed recursively: Traversing the query Q and structural tables we search for each interface of A_i and A_j, s.t. $A_i \neq A_j$.
   (a) In case of redundant intersection of A_i and A_j, for both A_i and A_j new outgoing edges are created.
   (b) In case of late binding influencing intersection of A_i and A_j, for both A_i and $A_i + A_j$ new outgoing edges are created.
   (c) In case of overriding intersection of A_i and A_j, a new outgoing edge for A_j is created.
4. The auxiliary node 1 and respective edges connecting all leaves of the graph with node 1 are created.

For the purpose of searching for the best evaluation sequence of storage strategies, each edge $e_{ij} \in G^{eval}$ is assigned its *length* which expresses the cost $cost_{eval}(Q_i, A_j)$ of evaluating of a query part Q_i using a strategy A_j and cost $cost_{inter}(A_{prev}, A_j)$ of the interface between strategy A_{prev} used for evaluation of Q_{i-1} and current strategy A_j.

Definition 33 Length of edge $e_{ij} = \langle v_x, v_y \rangle$ *of evaluation graph* G^{eval} *is defined as follows:*

$$length(e_{ij}) = \begin{cases} cost_{eval}(Q_i, A_j) & v_x = 0 \\ cost_{eval}(Q_i, A_j) + cost_{inter}(A_{prev}, A_j) & v_x \neq 0 \\ 0 & v_y = 1 \end{cases}$$

Now, having a graph G^{eval} and corresponding lengths of its edges, the problem of finding the optimal evaluation sequence of strategies transforms to the shortest path problem, i.e. searching the shortest path from node 0 to 1, which can be solved, e.g., using the classical Dijkstra's algorithm.

Reconstruction of XML Fragments

Similarly to query evaluation also in case of reconstruction of resulting XML fragments there can occur multiple ways of retrieval of the relevant data. Consider the situation depicted in Figure 7.8, where element `Actor` is associated with annotation `GENERIC="edge"` and element `Filmography` with late binding influencing annotation `INTERVAL="true"`, and query

```
/Actor/Filmography/Movie[Year=2007]
```

whose SQL translation returns a set $R = \{r_1, r_2, ..., r_k\}$ containing values of `recordID` column of records from table `EdgeFilmography` which fulfill the query. The required XML result is a set of elements `Movie`, each containing subelements `Title` and `Year` with corresponding values. With regard to the specified annotations we have two possibilities how to retrieve corresponding information – the Edge mapping or the Interval encoding.

In the former case we process each `recordID` $r_i \in R$; $i = 1, 2, ..., k$ separately: Firstly, we create a new `Movie` node of DOM tree T_{DOM} of the XML result. Then we select the set of all its subelements (and attributes) from the `EdgeFilmography` table. Subelements having a text content are added to the result, i.e. for each element a corresponding node in T_{DOM} is created. Subelements having element content are processed recursively. In general, for the purpose of the reconstruction we need to perform $O(k \cdot n)$ select queries from the `EdgeFilmography` table, where n is the maximum number of non-leaf nodes of XML fragments rooted at element `Movie`.

In the latter case, i.e. when exploiting the Interval encoding, the retrieval of the relevant information is much easier. The Interval encoding enables to retrieve information of the whole XML fragment at once and totally ordered. Having the set $R = \{r_1, r_2, ..., r_k\}$ we need a single query which contains a single join of two tables `EdgeFilmography`:

```
SELECT *
FROM EdgeFilmography m, EdgeFilmography e
WHERE m.recordID IN (r1, r2, ..., rk) AND
      m.intervalStart <= e.intervalStart AND
      e.IntervalEnd   <= m.intervalEnd
ORDER BY e.intervalStart
```

Thus, also in case of document reconstruction we need to choose the most efficient way of retrieval of the data. For this purpose we can use the same

approach as in case of query evaluation. The only difference is, that the costs of the strategies can differ. As for our sample set of annotations the most striking example is the `TOCLOB` storage strategy, where in case of query evaluation it has a high cost assuming that it requires preprocessing of the CLOB content, whereas in case of reconstruction its cost is low.

7.4 Architecture of the System

As depicted in Figure 7.13 the architecture of experimental system *UserMap* consists of several modules which can be divided according to phases of processing they belong to. The particular modularity is given not only by logical partitioning of the system, but it is also influenced by the needs of experiments and corresponding ability to omit various modules.

Phase I. Preparation

Being given an annotated XSD schema S the system first checks its validity using the *Xerces Java parser* [59] and builds its DOM tree T_S [17]. Then, for easier processing, the DOM tree is transformed into a *DOM graph* G_S [49], i.e. a graph representation of XSD schema similar to DTD graph (see Definition 5) extended for XML Schema constructs. For the same purpose the *graph builder* extracts the set of annotated fragments F_{orig}.

Phase II. Searching for Annotation Candidates

At this phase of the processing the *BAS module* performs the BAS algorithm (see Algorithm 4) and then the *GAS module* performs the GAS algorithm (see Algorithm 5) being given the schema S and the set of user-specified annotations F_{orig}. The former module identifies the set of annotation candidates F_{BAS}, the latter one the set of annotation candidates F_{GAS}. This way the system enables to skip any of the approaches, to compare their results, and thus to compare the efficiency of the resulting storage strategies. The resulting set of annotation candidates $F_{adapt} = F_{BAS} \cup F_{GAS}$.

We also assume that the similarity measure exploited in both the approaches is tuned according to the approach specified in Chapter 5. But since the *similarity evaluator* is a separate module evaluating similarity of two given schema fragments f and g, it can be easily replaced with any other method.

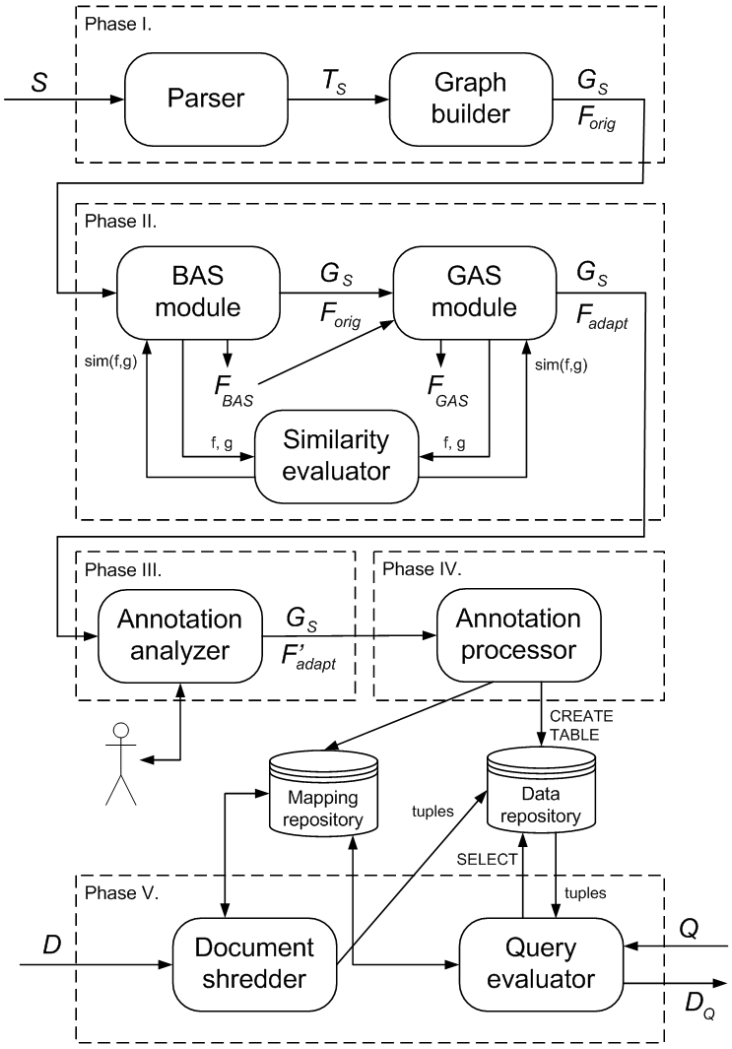


Figure 7.13: Architecture of the system

Phase III. Correction of Candidate Set

At this phase the candidate set of annotations F_{adapt} needs to be corrected. As described in Chapter 7.2 there are two types of correction – automatic and user-specified – resulting in set of corrected annotations F'_{adapt}. In the former case the *annotation analyzer* automatically searches and removes the forbidden annotations, in the latter case a user interaction is required. In the experimental implementation of the system the user interaction is omitted and a default possibility is always applied. But as mentioned in the Conclusion, the very next enhancement of the system will focus on user interaction, user-friendliness, and appropriate GUI. Then also the incorrect situations can be identified and solved using the user interaction as well.

Phase IV. XML-to-Relational Mapping

At this phase the *annotation processor* parses the set of corrected annotations and maps the corresponding schema fragments into the *data repository* using the respective approaches. As described in Chapter 7.3.1, at the same time the system stores the information about schema mapping into supplemental structural tables in the *mapping repository*.

Phase V. Document Shredding and Query Evaluation

At this phase the system is ready for the intended application. It involves two operations:

- shredding a document D valid against XML schema S into corresponding tables and
- evaluation of query Q posed over the schema S which returns document D_Q containing corresponding results.

The *document shredder* reads the XML document using a SAX parser and on the basis of the information from mapping repository generates an SQL script for storing appropriate tuples into the data repository. The *query evaluator* firstly identifies the most efficient evaluation sequence (as described in Chapter 7.3.4) and then, also using information from mapping repository, translates the XML query into an SQL query. Similarly, the resulting tuples are then transformed to corresponding XML fragments using the most efficient reconstruction sequence.

7.5 Conclusion

As can be seen, there are several interesting issues related to the relatively simple idea of hybrid user-driven mapping strategy. In this chapter we have outlined the key components of the whole system and using simple examples illustrated the related problems. As it is obvious, most of the approaches (such as, e.g., efficient query translation, cost estimation of the queries, etc.) can be significantly optimized, since a detailed research has already been done in these areas. But this is already out of the scope of this thesis and remains in the list of future work.

Chapter 8

Conclusion

The main aim of this thesis 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 analyzed the disadvantages and weak points of currently most efficient methods of this type, so-called adaptive ones, and proposed their enhancing – a hybrid user-driven adaptive XML-to-relational mapping algorithm. This approach is able to exploit the user given information, i.e. schema annotations, more deeply and, at the same time, to propose more appropriate candidate mapping strategy for the not annotated schema parts using an adaptive approach. For the purpose of the method we have proposed a similarity measure focussing on structural similarity of the data and an algorithm for realistic tuning of its parameters. Most parts of the proposal (such as, e.g., the tuning process or the experimental tests) were based on the results of statistical analysis of real-world XML data, i.e. a reasonable real source of information. Finally, we have dealt with the correctness and structure of the resulting storage strategy and the key aspects of related query evaluation. In particular we have focussed on evaluation of parts of a single XML query using various storage strategies and exploitation of redundancy.

The very next step of our future work is an elaborate implementation of the proposed system with the emphasis on all the "side" aspects of the proposal including the omitted user-friendly interface which is definitely and important requirement for a system based on user interaction. During the research experimental and prototype implementations of the most important modules of the system were created for the purpose of evaluation or verification of important algorithms. But at this stage we intend to implement

the system as a complete and robust application, similarly to systems described in [34] or [23]. And this plan is also closely related to the mentioned open issue of optimization of the query evaluator. Similarly to paper [36] we intend to exploit a cost evaluator which is able to dynamically adapt the statistics to the newly coming data and, at the same time, to conform to the assumption of multiple storage strategies used within a single schema.

As mentioned in Chapter 3 there are several further possible ways of improving XML data management using (O)RDBMS. Probably the most important as well as difficult one is the dynamic adaptability, i.e. the ability to adapt the relational schema to the changing requirements. Such system would solve the most striking disadvantage of current adaptive methods – the fact that the target schema is adapted only to a single initial application. Therefore if the data or queries slightly change, the efficiency of processing can significantly decrease, sometimes even below the efficiency of a general fixed method. And it is obvious that such changes of the application are very probable. On the other hand, we can expect that though the changes can occur, probably they will not be radical. Thus also the resulting changes of the relational schema will be rather local. With this assumption a research can be done on the impact of such local changes on the total efficiency of query evaluation, types of such changes and their consequences, the optimal frequency of schema adaptation, etc.

A first step towards the dynamic adaptability could be a combination of the proposed system with a classical adaptive strategy. We could assume that the user provides not only a set of annotations but also sample XML documents and XML queries. Then we can use a combination of the annotation-based and classical adaptive method. We can naturally expect that such combination will result in more efficient storage strategy, especially with respect to the provided queries. Nevertheless, the key shortcoming of such approach is evident – the user is expected to provide too many information. But assuming that the sample queries and data can be gathered dynamically, also this disadvantage disappears.

As we have mentioned in the Introduction, the scientific research has already proven that management of XML data using (O)RDBMS is possible, but undoubtedly less efficient than the native approaches. As we have discussed, since there is currently no robust and verified implementation of a native XML database which could compete with database systems such as Oracle Database [14], IBM DB2 [11], Microsoft SQL Server [12], etc., the research focussing on further improvements of XML processing based on

(object-)relational databases is still important and necessary. On the other hand, the question is how long such research should be maintained. From one point of view we could say that until the native methods reach the necessary level of reliability, i.e. maybe few more years. But, on the other hand, we can ask whether the native databases will be able to reach the popularity of classical relational database systems. The history has already showed that the practical world prefers simple and easy-to-learn solutions and this requirement the basic ideas of relational model satisfy perfectly. The question is whether we will not witness the same situation as in case of object-oriented databases which were not able to compete with this advantage.

From another point of view we can observe that the requirement of managing various kinds of data in the same repository is not of marginal importance. Therefore all the database vendors extend their system enabling to store and process, e.g., bitmaps, multimedia data, multidimensional vectors, meta data, etc. Thus another question is whether it is reasonable to focus on database systems which are able to manage only a single XML data format.

We can find many arguments for and against the possible answer to the above open issues. But only the future will show which approach is more suitable for practical exploitation that is the most reasonable metric of meaningfulness of any research.

Bibliography

[1] Available at: http://www.ibiblio.org/bosak/.

[2] Available at: http://monetdb.cwi.nl/xml/index.html.

[3] Available at: http://inex.is.informatik.uni-duisburg.de:2004/.

[4] Available at: http://www.freedb.org/.

[5] Available at: http://www.cs.wisc.edu/niagara/data.html.

[6] Available at: http://www.nlm.nih.gov/mesh/meshhome.html.

[7] Available at: http://arthursclassicnovels.com/.

[8] Available at: http://research.imb.uq.edu.au/rnadb/.

[9] *DB2 XML Extender*. IBM. http://www.ibm.com/.

[10] *DocBook Technical Committee Document Repository*. OASIS. http://www.oasis-open.org/docbook/.

[11] *IBM DB2*. IBM. http://www-306.ibm.com/software/data/db2/.

[12] *Microsoft SQL Server*. Microsoft Corporation. http://www.microsoft.com/sql/default.mspx.

[13] *OpenOffice.org Project*. Sun Microsystems. http://www.openoffice.org/.

[14] *Oracle Database*. Oracle Corporation. http://www.oracle.com/database/index.html.

[15] *Oracle XML DB*. Oracle Corporation. http://www.oracle.com/.

[16] *The Semantic Web Homepage*. W3C. `http://www.w3.org/2001/sw/`.

[17] *Document Object Model (DOM)*. W3C, 2005. `http://www.w3.org/DOM/`.

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

[19] *Scalable Vector Graphics (SVG) 1.1 Specification*. W3C, January 2003. `http://www.w3.org/TR/SVG/`.

[20] M. Altinel and M. J. Franklin. Efficient Filtering of XML Documents for Selective Dissemination of Information. In *VLDB '00: Proc. of the 26th Int. Conf. on Very Large Data Bases*, pages 53–64, San Francisco, CA, USA, 2000. Morgan Kaufmann Publishers Inc.

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

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

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

[24] D. Beckett. *RDF/XML Syntax Specification (Revised)*. W3C, February 2004. `http://www.w3.org/TR/rdf-syntax-grammar/`.

[25] 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.

[26] G. J. Bex, F. Neven, and J. Van den Bussche. DTDs versus XML Schema: a Practical Study. In *WebDB '04: Proc. of the 7th Int. Workshop on the Web and Databases*, pages 79–84, New York, NY, USA, 2004. ACM Press.

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

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

[29] B. Choi. What Are Real DTDs Like? In *WebDB '02: Proc. of the 5th Int. Workshop on the Web and Databases*, pages 43–48, Madison, Wisconsin, USA, 2002. ACM Press.

[30] I. Choi, B. Moon, and H.-J. Kim. A Clustering Method Based on Path Similarities of XML Data. *Data Knowl. Eng.*, 60(2):361–376, 2007.

[31] U. Chukmol, R. Rifaieh, and A.-N. Benharkat. EXSMAL: EDI/XML Semi-Automatic Schema Matching Algorithm. In *CEC '05: Proc. of the 7th IEEE Int. Conf. on E-Commerce Technology*, pages 422–425, Munchen, Germany, 2005. IEEE Computer Society.

[32] D. DeHaan, D. Toman, M. P. Consens, and M. T. Ozsu. A Comprehensive XQuery to SQL Translation Using Dynamic Interval Encoding. In *SIGMOD '03: Proc. of the 22nd ACM SIGMOD Int. Conf. on Management of Data*, pages 623–634, New York, NY, USA, 2003. ACM Press.

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

[34] F. Du, S. Amer-Yahia, and J. Freire. ShreX: Managing XML Documents in Relational Databases. In *VLDB '04: Proc. of the 30th Int. Conf. on Very Large Data Bases*, pages 1297–1300, Toronto, ON, Canada, 2004. Morgan Kaufmann Publishers Inc.

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

[36] J. Freire, J. R. Haritsa, M. Ramanath, P. Roy, and J. Simeon. StatiX: Making XML Count. In *SIGMOD '02: Proc. of the 21st ACM SIGMOD Int. Conf. on Management of Data*, pages 181–192, Madison, Wisconsin, USA, 2002. ACM Press.

[37] J. H. Holland. *Adaptation in Natural and Artifical Systems.* University of Michigan Press, Ann Arbor, MI, USA, 1975.

[38] S. Kirkpatrick, C. D. Gerlatt Jr., and M.P. Vecchi. Optimization by Simulated Annealing. *Science*, 220(4598):671–680, 1983.

[39] M. Klettke and H. Meyer. XML and Object-Relational Database Systems – Enhancing Structural Mappings Based on Statistics. In *Selected papers from the 3rd Int. Workshop WebDB '00 on The World Wide Web and Databases*, pages 151–170, London, UK, 2001. Springer-Verlag.

[40] M. Klettke, L. Schneider, and A. Heuer. Metrics for XML Document Collections. In *XMLDM Workshop*, pages 162–176, Prague, Czech Republic, 2002.

[41] R. Krishnamurthy, V. Chakaravarthy, and J. Naughton. On the Difficulty of Finding Optimal Relational Decompositions for XML Workloads: A Complexity Theoretic Perspective. In *ICDT '03: Proc. of the 9th Int. Conf. on Database Theory*, pages 270–284, Siena, Italy, 2003. Springer.

[42] R. Krishnamurthy, R. Kaushik, and J. Naughton. XML-to-SQL Query Translation Literature: The State of the Art and Open Problems. In *XSym '03: Proc. of the 1st Int. XML Database Symp.*, volume 2824, pages 1–18, Berlin, Germany, 2003. Springer.

[43] A. Kuckelberg and R. Krieger. Efficient Structure Oriented Storage of XML Documents Using ORDBMS. In *Proc. of the VLDB '02 Workshop EEXTT and CAiSE '02 Workshop DTWeb*, pages 131–143, London, UK, 2003. Springer-Verlag.

[44] D. Lee and W. W. Chu. Constraints-Preserving Transformation from XML Document Type Definition to Relational Schema. In *ER '00: Proc. of the 19th Int. Conf. on Conceptual Modeling*, volume 1920 of *Lecture Notes in Computer Science*, pages 323–338. Springer, 2000.

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

[46] A. McDowell, C. Schmidt, and K. Yue. Analysis and Metrics of XML Schema. In *SERP '04: Proc. of the Int. Conf. on Software Engineering,*

Research and Practice, pages 538–544, Las Vegas, Nevada, USA, 2004. CSREA Press.

[47] L. Mignet, D. Barbosa, and P. Veltri. The XML Web: a First Study. In *WWW '03: Proc. of the 12th Int. Conf. on World Wide Web, Volume 2*, pages 500–510, New York, NY, USA, 2003. ACM Press.

[48] T. Milo and S. Zohar. Using Schema Matching to Simplify Heterogeneous Data Translation. In *VLDB '98: Proc. of the 24th Int. Conf. on Very Large Data Bases*, pages 122–133, San Francisco, CA, USA, 1998. Morgan Kaufmann Publishers Inc.

[49] I. Mlynkova and J. Pokorny. From XML Schema to Object-Relational Database – an XML Schema-Driven Mapping Algorithm. In *ICWI '04: Proc. of the 3rd IADIS Int. Conf. WWW/Internet*, pages 115–122, Madrid, Spain, 2004. International Association for Development of the Information Society.

[50] I. Mlynkova and J. Pokorny. XML in the World of (Object-)Relational Database Systems. In *ISD '04: Proc. of the 13th Int. Conf. on Information Systems Development*, pages 63–76, Vilnius, Lithuania, 2004. Springer Science+Business Media Inc.

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

[52] I. Mlynkova, K. Toman, and J. Pokorny. *Statistical Analysis of Real XML Data Collections*. Technical report 2006/5, Charles University, Prague, Czech Republic, June 2006.

[53] C.-H. Moh, E.-P. Lim, and W. K. Ng. DTD-Miner: A Tool for Mining DTD from XML Documents. In *WECWIS '00: Proc. of the 2nd Int. Workshop on Advanced Issues of E-Commerce and Web-Based Information Systems*, pages 144–151, Milpitas, CA, USA, 2000. IEEE Computer Society.

[54] M. Murata, D. Lee, and M. Mani. Taxonomy of XML Schema Languages Using Formal Language Theory. *ACM Trans. Inter. Tech.*, 5(4):660–704, 2005.

[55] S. Nestorov, S. Abiteboul, and R. Motwani. Extracting Schema from Semistructured Data. In *SIGMOD '98: Proc. of the 17th ACM SIGMOD Int. Conf. on Management of Data*, pages 295–306, Seattle, Washington, DC, USA, 1998. ACM Press.

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

[57] A. Nierman and H. V. Jagadish. Evaluating Structural Similarity in XML Documents. In *WebDB '02: Proc. of the 5th Int. Workshop on the Web and Databases*, pages 61–66, Madison, Wisconsin, USA, 2002. ACM Press.

[58] J. Pokorny. XML Data Warehouse: Possibilities and Solutions. In *Constructing the Infrastructure for the Knowledge Economy: Methods & Tools, Theory & Practice*, pages 531–542, Dordrecht, The Netherlands, 2004. Kluwer Academic Publishers.

[59] The Apache XML Project. *Xerces Java Parser 1.4.4*. The Apache Software Foundation, 2005. `http://xerces.apache.org/xerces-j/`.

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

[61] E. Rahm and T. Bohme. *XMach-1: A Benchmark for XML Data Management*. Database Group Leipzig, 2006.

[62] M. Ramanath, J. Freire, J. Haritsa, and P. Roy. Searching for Efficient XML-to-Relational Mappings. In *XSym '03: Proc. of the 1st Int. XML Database Symp.*, volume 2824, pages 19–36, Berlin, Germany, 2003. Springer.

[63] N. Rishe, J. Yuan, R. Athauda, S.-C. Chen, X. Lu, X. Ma, A. Vaschillo, A. Shaposhnikov, and D. Vasilevsky. Semantic Access: Semantic Interface for Querying Databases. In *VLDB '00: Proc. of the 26th Int. Conf. on Very Large Data Bases*, pages 591–594, San Francisco, CA, USA, 2000. Morgan Kaufmann Publishers Inc.

[64] K. Runapongsa and J. M. Patel. Storing and Querying XML Data in Object-Relational DBMSs. In *EDBT '02: Proc. of the Worshops*

XMLDM, MDDE, and YRWS on XML-Based Data Management and Multimedia Engineering-Revised Papers, pages 266–285, London, UK, 2002. Springer-Verlag.

[65] A. Sahuguet. Everything You Ever Wanted to Know About DTDs, But Were Afraid to Ask (Extended Abstract). In *WebDB '00: Selected Papers from the 3rd Int. Workshop on The World Wide Web and Databases*, pages 171–183, London, UK, 2001. Springer-Verlag.

[66] A. Schmidt, M. L. Kersten, M. Windhouwer, and F. Waas. Efficient Relational Storage and Retrieval of XML Documents. In *Selected papers from the 3rd Int. Workshop WebDB '00 on The World Wide Web and Databases*, pages 137–150, London, UK, 2001. Springer-Verlag.

[67] 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: Proc. of the 25th Int. Conf. on Very Large Data Bases*, pages 302–314, San Francisco, CA, USA, 1999. Morgan Kaufmann Publishers Inc.

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

[69] 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: Proc. of the 21st ACM SIGMOD Int. Conf. on Management of Data*, pages 204–215, Madison, Wisconsin, USA, 2002. ACM Press.

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

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

[72] 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.

[73] M. Yoshikawa, T. Amagasa, T. Shimura, and S. Uemura. XRel: A Path-Based Approach to Storage and Retrieval of XML Documents Using Relational Databases. *ACM Trans. Inter. Tech.*, 1(1):110–141, 2001.

[74] Z. Zhang, R. Li, S. Cao, and Y. Zhu. Similarity Metric for XML Documents. In *FGWM '03: Proc. of Workshop on Knowledge and Experience Management*, Karlsruhe, Germany, 2003.

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