Adaptive XML-to-Relational Storage Strategies

Irena Mlynkova
Department of Software Engineering
School of Computer Science
Faculty of Mathematics and Physics
Charles University
Malostranske nam. 25
118 00 Praha 1
Czech Republic

Tel: +420-221-914-264
Fax: +420-221-914-323
E-mail: irena.mlynkova@mff.cuni.cz
INTRODUCTION

Without any doubt, the eXtensible Markup Language (XML) (Bray et al., 2006) is currently one of the most popular formats for data representation. Its wide popularity naturally invoked an enormous endeavour to propose faster and more efficient methods and tools for managing and processing of XML data. Soon it was possible to distinguish several different directions. The four most popular ones are methods which store XML data in a classical file system, methods which store and process XML data using a relational database management system, methods which exploit a pure object-oriented approach and native methods that use special indices, numbering schemas and/or data structures proposed or suitable particularly for tree structure of XML data.

Naturally, each of these approaches has particular advantages or disadvantages. In general, especially the popularity of file system-based and pure object-oriented methods is low. The former ones suffer from inability of querying without any additional pre-processing of the data, whereas the latter approach fails especially in finding a corresponding efficient and comprehensive tool. 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. Nevertheless, the most practically used methods exploit features of relational databases. Although the scientific world has already proven that native XML strategies perform much better, they still lack one important aspect – a reliable and robust implementation verified by years of both theoretical and practical effort.

Currently there exists a plenty of existing works concerning database-based XML data management. All the major database vendors support XML and even the SQL (Structured Query Language) standard has been extended by a new part SQL/XML (ISO/IEC 9075-14, 2003) which introduces new XML data type and operations for XML data manipulation. But, although the amount of existing works is enormous, there is probably no universally efficient strategy. Each of the existing approaches is suitable for selected applications, but, at the same time, there can usually be found cases when it can be highly inefficient. An illustrative example is updatability of data, where the efficient storage strategies significantly differ if the feature is required or not.

---

1 In the rest of the text the term “database” represents a relational database management system (RDBMS).
The aim of this text is to provide an overview of existing XML-to-relational storage strategies. We will overview their historical development and provide a more detailed discussion of the currently most promising ones – the adaptive methods. Finally, we will outline possible future directions.

BACKGROUND
The main concern of the database-based XML techniques is the choice of the way XML data are stored into relations, so-called XML-to-relational mapping or schema decomposition to relations. The strategy in first approaches to XML-to-relational mapping, so-called generic (e.g. Florescu et al., 1999), was based purely on the data model of XML documents. The methods were able to store any kind of XML data since they viewed XML documents as general labelled trees. But, the efficiency of query evaluation was quite low due to numerous join operations or the increase of efficiency was gained at the cost of increase of space overhead.

Hereafter, the scientists came with a natural idea to exploit structural information extracted from XML schemas of XML data, usually expressed in DTD (Document Type Description) (Bray et al., 2006) or XML Schema (Thompson et al., 2004; Biron et al., 2004) language. All the so-called schema-driven approaches (e.g. Shanmugasundaram et al., 1999) were based on the same idea that the structure of the target relational schema can be created according the structure of the source XML schema. Assuming that a user specifies the XML schema as precisely as possible to specify the related data, we can get also more precise relational XML schema. The problem is that DTDs are usually too general. The extensive examples are recursion or * operator which, in general, enable to specify infinitely deep or wide XML documents. According to analyses of real-world XML data (Mlynkova et al., 2006) in both the cases the respective XML documents are much simpler and, thus, the effort spent on processing all the complex schema constructs is useless.

A slight solution to the problem brought schema-driven approaches exploiting information extracted from XML Schema definitions (e.g. Mlynkova et al., 2004). The XML Schema language enables to describe the structure of XML data more precisely, especially in case of data types. But, although its expressive power is higher, most of the
new constructs can be considered as “syntactic sugar”. Hence, exploitation of XML Schema constructs does not have a key impact on efficiency of XML processing.

A different type of improving of the fixed mapping strategies brought constraint-preserving methods (e.g. Chen et al., 2003; Davidson et al., 2007) which focus on preservation of constraints specified in the source XML schema, such as, e.g., key and foreign key constraints, functional dependencies or semantic constraints, in the target relational schema. Such approaches enable to reduce redundancy as well as improve efficiency of selected queries and, especially, update operations. But, in general, the methods suffer from the same disadvantages as all the described methods.

Consequently, it became obvious that a fixed universally efficient approach does not exist and to achieve a higher performance it is necessary to adapt the relational schema to requirements of the current application. Consequently, the adaptive methods have appeared and they are still considered as the most efficient approaches to the XML-to-relational mapping. The first and natural idea of adaptability was to leave all the decisions in hands of a user who specifies both the target relational schema and the required mapping. We speak about so-called user-defined methods (see Amer-Yahia, 2003). But the problem is that these methods require a user mastering two complex technologies. Consequently, two different types of approaches have appeared. In case of user-driven methods (Amer-Yahia et al., 2004; Balmin et al., 2005; Mlynkova, 2007) a user is provided with a fixed mapping strategy and can specify just local changes where appropriate. On the other hand, cost-driven approaches (Klettke et al., 2000; Ramanath et al., 2003; Xiao-ling et al., 2003; Zheng et al., 2003) search a space of possible XML-to-relational mapping strategies and choose the one which conforms to the current application the most. The application is usually specified using a sample set of XML documents and XML queries.

**ADAPTIVE APPROACHES**

As we have described, the existing representatives of adaptive methods can be divided into cost-driven and user-driven. Both approaches 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, in the latter case it is specified by user-given annotations, i.e. the user
directly specifies the required changes of a default mapping.

**Cost-Driven Techniques**

Cost-driven techniques can choose the best storage strategy for a particular application automatically, without any interference of a user. Apart from various parameters of particular algorithms, the user can influence the mapping process through the provided XML schema, set of sample XML documents and XML queries. But after providing the input information, the algorithms cannot be influenced further.

We can divide the existing cost-driven approaches into two types – *single-candidate* and *multiple-candidate*. Single-candidate approaches involve a straightforward mapping algorithm which provides a single output relational schema on the basis of input data. Multiple-candidate approaches also process the input data, but before providing the resulting relational schema, they evaluate multiple candidate solutions and choose the one which suits the considered application the most.

One of the first attempts of a cost-driven adaptive approach and, at the same time, probably the only known representative of single-candidate approaches is a method called *Hybrid object-relational mapping* (Klettke et al., 2000). It is based on the observation that if XML documents are mostly semi-structured, a classical decomposition of unstructured or semi-structured XML parts into relations leads to inefficient query processing caused by plenty of join operations. Hence, the algorithm stores well structured parts into relations and semi-structured parts using an *XML data type*, which supports path queries and XML-aware full-text operations. The main concern is to identify the structured and semi-structured parts of the input XML schema, whereas the proposed method is based on the analysis of their amount and way of occurrence in the sample XML documents and XML queries.

On the other hand, each of the existing multiple-candidate techniques (Ramanath et al., 2003; Xiao-ling et al., 2003; Zheng et al., 2003) can be characterized by:

- an initial input XML schema \( S_{init} \),
- a set of XML schema transformations \( T = \{t_1, t_2, ..., t_n\} \), where \( \forall i : t_i \) transforms a given schema \( S \) into a schema \( S' \),
- a fixed XML-to-relational mapping function \( f_{map} \) which transforms a given XML schema \( S \) into a relational schema \( R \),
• a set of input sample data $D_{sample}$ consisting of a set of sample XML documents $D$ and XML queries $Q$ which characterize the future application and
• a cost function $f_{cost}$ which evaluates the efficiency of a given relational schema $R$ with regard to the set $D_{sample}$.

Examples of schema transformations can be \textit{inlining/outlining} of an element into/out of the table of its parent element, or storing a whole schema fragment into a single BLOB column.

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 is based on the idea of “brute force”.

It first generates a set of possible XML schemas $\Sigma$ using transformations from set $T$ and starting from initial schema $S_{init}$. Then it searches for schema $s \in \Sigma$ with minimal cost $f_{cost}(f_{map}(s), D_{sample})$ and returns the corresponding optimal relational schema $R_{opt} = f_{map}(s)$.

It is obvious that the complexity of such algorithm strongly depends on the set $T$. It can be proven that even a simple set of transformations causes the problem of finding the optimal schema to be NP-hard (Xiao-ling et al., 2003). Thus, in fact, the existing techniques search for a suboptimal solution using various heuristics, greedy strategies, approximation algorithms, terminal conditions etc.

\textbf{User-Driven Techniques}

Undoubtedly, the most flexible adaptive approaches to XML-to-relational mapping are user-defined ones which leave the whole process in hands of a user who defines both the target database schema and the required mapping. Probably due to simple implementation they are especially popular and supported in most commercial database systems. The problem is that such approach assumes users skilled in two complex technologies – relational databases and XML. Furthermore, for more complex applications the design of an optimal relational schema is generally not an easy task.

The main difference of user-driven methods is that the user can influence a default fixed mapping strategy using annotations which specify the required mapping for particular schema fragments. The set of allowed mappings is naturally limited, but still enough powerful to define various mapping strategies. In other words, the user helps the mapping process, he/she does not perform it directly.
Each of the techniques is characterized by:

- an initial XML schema $S_{init}$,
- a set of fixed XML-to-relational mappings $f_{map}^1, f_{map}^2, \ldots, f_{map}^n$,
- a set of annotations $A$, each of which is specified by name, target, allowed values and function and
- a default mapping strategy $f_{def}$ for not annotated fragments.

Examples of annotations are usually similar to schema transformations of cost-driven strategies. But, in general, an annotation can specify any kind of a fixed storage strategy. The existing approaches can be also divided according to strategy used to provide the resulting relational schema. We differentiate so-called direct and indirect mapping.

The approaches exploiting direct mapping (Amer-Yahia et al., 2004; Balmin et al., 2005) are based on a straightforward algorithm. Firstly, the initial schema $S_{init}$ is annotated with user-required mapping strategies from $A$. Secondly, the correctness of the annotations is checked. And finally, the XML schema is mapped – for annotated schema fragments the specified mapping strategies $f_{map}^1, f_{map}^2, \ldots, f_{map}^n$ are used, for not annotated schema fragments $f_{def}$ is used. The existing approaches differentiate mainly in the supported mapping strategies.

Indirect mapping strategies try to exploit the user-provided information as much as possible. They are based on the idea that the annotations can not only be directly applied on particular schema fragments, but the information can be exploited for mapping the remaining schema fragments. The first and probably still the only representative of indirect user-driven strategies is system UserMap (Mlynkova, 2007). The indirect mapping proposed in the system is based on a simple observation that the user-provided schema annotations can be viewed as hints how to store particular schema fragments. And if we know the required mapping strategy for a particular schema fragment, we can assume that structurally (or semantically) similar schema fragments should be stored in a similar way. Hence, the system iteratively searches for similar schema fragments to find more suitable mapping strategy.

**FUTURE TRENDS**

Although each of the existing approaches brings certain interesting ideas and optimizations, there is still a space of possible future improvements. Apart from minor
open issues, such as speeding up the search strategy of cost-driven methods, simplifying user interaction of user-driven strategies or possibility of combining of the approaches, the most striking disadvantage of adaptive methods is the problem of possible changes of both XML queries and XML data that can lead to crucial worsening of their efficiency. The solution to this problem – i.e. a system that is able to adapt dynamically – is obvious and challenging, but it is not an easy task. The first related problem is how to find the new mapping strategy efficiently. Naturally, we could repeat the whole search strategy after a certain amount of operations over the current relational schema has been performed, but this naive strategy would be quite inefficient. A better solution seems to be exploitation of a search strategy that does not have to be restarted when the related information change, i.e. a strategy which can be applied on dynamic systems, such as, e.g., ACO (Ant Colony Optimization) meta-heuristic (Dorigo et al., 2006). The dynamic system should also avoid total reconstructions of the whole relational schema and corresponding necessary reinserting of all the stored data or such operation should be done only in very special cases and not often. On the other hand, this “brute-force” approach can serve as a good inspiration. It is possible to suppose that changes especially in case of XML queries will not be radical, but will have a gradual progress. Thus the changes of the relational schema will be mostly local and we can apply the expensive reconstruction just locally. Furthermore, we can again exploit the idea of pattern matching and try to find the XML pattern defined by the modified schema fragment in the rest of the schema.

Another question is how often should be the relational schema reconstructed. The natural idea is of course “not too often”. But, 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.

**CONCLUSION**

The main goal of this text was to describe and discuss the current state of the art and open issues of database-based XML-processing methods with the focus of the currently most efficient representatives, so-called adaptive approaches. First, we have provided a
motivation why this topic should be ever studied and why adaptive methods are so promising. Then we have provided an overview and classification of the existing approaches and their features. And, finally, we have discussed the corresponding open issues and their possible solutions.

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

BIBLIOGRAPHY


KEY TERMS

XML-to-relational mapping: A method which specifies how XML data are stored into relations of a relational database management system. It involves a definition of the target database schema and a way how the data are stored into its relations. Related problems are data retrieval and data updates, but they are usually determined directly by the storage strategy.
Generic XML-to-relational mapping: An XML-to-relational mapping which is based purely on a selected kind of data model of XML documents.

Schema-driven XML-to-relational mapping: An XML-to-relational mapping where the target relational schema is defined according to the structure of the source XML schema of XML data.

Fixed XML-to-relational mapping: An XML-to-relational mapping which is based on a fixed set of rules how to create the target database schema.

Adaptive XML-to-relational mapping: An XML-to-relational mapping which is based on exploitation of additional information, such as, e.g., sample XML documents, sample XML queries, user-specified requirements etc. and respectively adapts the target relational schema.

User-defined XML-to-relational mapping: An XML-to-relational mapping where the user specifies both the target relational schema and the required mapping manually.

User-driven XML-to-relational mapping: An XML-to-relational mapping where the user is provided with a fixed mapping strategy and can specify its local changes where appropriate.

Cost-driven XML-to-relational mapping: An XML-to-relational mapping which searches a space of possible fixed XML-to-relational mapping methods and chooses the one which conforms to the current application, i.e. XML documents and XML queries, the most.