Overview and Possible Improvements of Techniques for Processing XML Documents in (O)RDBMS

Irena Mlynkova

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

Abstract. XML is presently considered as one of the best standards for data representation. But with its growing usage the problem of effective processing of XML documents arises. A natural idea is to store and manage XML data in currently existing (O)RDBMS. At present there is a plenty of such techniques, each of which has its advantages and disadvantages, whereas the strongest objection against is that the database processing can be in certain cases quite slow.
This contribution contains an overview of existing techniques for processing XML documents in (O)RDBMS and corresponding mapping methods, a brief description of own formerly proposed mapping method and finally a proposal and a discussion of their possible future improvements and optimalizations.

1 Introduction

In recent years XML is considered as one of the best standards for data representation. Obviously, the main reason for this trend is that XML includes powerful tools for definition of the required structure of the data, which considerably simplifies their exchanging and processing.

On the contrary, the growing usage of XML technologies raises the problem of effective management of XML documents and querying the data. On one hand there is a set of powerful XML standards, but on the other hand it is necessary to solve the problem of their effective implementation.

A natural idea is to store and manage XML data in existing (object-)relational database systems ((O)RDBMS). This solution results from the way of describing the structure of XML documents, which resembles (object-)relational features and from other database features (e.g. query languages, schemes, etc.) XML technologies involve. What is more, combining these two technologies provides XML with missing database mechanisms (e.g. transactions, indexes, multi-user access, etc.).

At present, there is a plenty of such techniques, each of which has its advantages and disadvantages as well as common characteristics according to which they can be classified. Obviously the strongest objection against this idea is that the database processing can be in certain cases quite slow. Consequently, there are several important questions to be considered such as: Is it possible to speed up the processing of XML documents in (O)RDBMS? Where are the limits and borders of these improvements? Are the advantages of combining XML and database technologies worthwhile concerning the idea?

This paper is trying to deal with these points and issues. Section 2 contains an overview of existing techniques for processing XML documents in (O)RDBMS. Section 3 contains a classification of existing mapping methods and a brief description of own formerly proposed method. According to the classification Section 4 discusses and proposes their possible future optimalizations and poses another questions and opened issues and Section 5 provides conclusions.

2 Overview and Classification of Existing Techniques

The most general classification of existing techniques for processing XML data using (O)RDBMS is connected with the basic classification of XML documents into *document-centric* and *data-centric* [2]. Documents from the former group have generally irregular structure, the order of sibling elements is mainly significant and they usually contain mixed-content elements, comments, CDATA sections, etc. Documents from the latter group have in principle contradictory characteristics.

2.1 Techniques for Document-Centric XML Documents

Processing document-centric XML documents requires preserving their structure as a whole, in many cases including such details like, e.g., white spaces. In other words we speak about a good level of *round tripping* [2], i.e. the process of storing an XML document in a database system and retrieving preferably the same document back.

Document-centric XML documents are usually stored as a whole in one table column of a LOB data type or in *native XML databases* (NXDs), eventually in their special types such as *persistent DOMs* or *content management systems*. In the former case the processing is relatively fast but we loose the possibility to query the stored data effectively. In the latter case the processing is "native" which means that an NXD supports "natural" ways for accessing the stored XML data – e.g. XML query languages, DOM [8] or SAX [5] interfaces, etc. The main disadvantage is that as an NXD uses a certain strategy for storing and ordering XML data, performance problems can encounter in case of retrieving the data in any form other than that in which it is logically stored.

2.2 Techniques for Data-Centric XML Documents

Techniques for data-centric XML documents have one common idea: XML data is stored and processed in an (O)RDBMS and using a certain method (so-called *mapping method* – see Section 3) transferred between relations and XML documents and vice versa. The level of round tripping can be usually low – it is necessary to preserve purely the elements, attributes, their hierarchical structure and the data itself. The transferring process can be provided either by a third party software, so-called *middleware*, or by the database itself – in this case we speak about *XML-enabled databases*.

A special kind of mapping methods is so-called *XML data binding technology*. It is based on the idea of mapping XML data to classes and objects of an object-oriented programming language (usually Java or C++) and enables applications to work with XML data using structures, which can be more suitable than, e.g., structures of a DOM tree.

In the following text we will focus mainly on data-centric methods and corresponding mapping algorithms.

3 Overview of Mapping Methods

As mentioned before mapping methods are methods for transferring XML data between XML documents and (object-)relational structures. At present there is a considerable number of these techniques, each of which has its advantages and disadvantages and common features according to which they can be classified. The basic classification includes following three classes:

– *generic methods*, which do not use any schema of stored XML documents,
– *schema-driven methods*, which are based on existing schema of stored XML documents, and
– *user-defined methods*, which are based on user-defined mapping.

This section contains a brief description and classification of these methods. A more comprehensive discussion can be found in [7].

3.1 Generic Mapping Methods

Generic mapping methods do not use (possibly) existing XML schema of stored XML documents. They are usually based on one of these approaches:

– to create a general (object-)relational schema into whose relations any XML document regardless its structure can be stored, or
– to create a special kind of (object-)relational schema into whose relations only a certain collection of XML documents having a similar structure can be stored.

The former methods model an XML document as a tree T according to e.g. the OEM model or the DOM model, while the latter reflect its special “relational” structure.

Generic-Tree Mapping. A typical representative of generic mapping methods is a group of methods called *Generic-tree mapping* [3]. An example of an XML document and its tree *T* is depicted in Fig. 1.

```
<person id=1 age=23>
   <name>Irena</name>
   <surname>Mlýnková</surname>
   <address id=2>
      <street>Podlesí 4943</street>
      <city>Zlín</city>
   </address>
</person>
<person id=3 age=30>
   <name>Jim</name>
   <surname>Beam</surname>
</person>
...
```

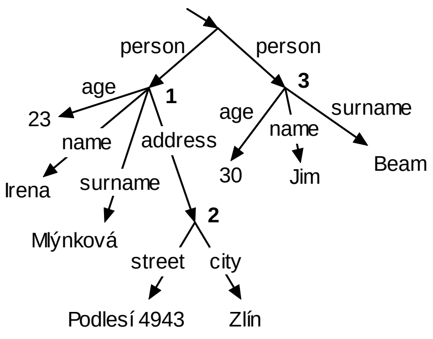

Fig. 1. An example of a generic-tree

Edge Mapping. This method stores all edges of *T* in the following table:

`Edge(source, ord, name, flag, targ)`

The table contains identifiers of nodes connected by the edge (`source` and `targ`), name of the edge (`name`), a flag that indicates whether the edge is internal or points to a leaf (`flag`), and an ordinal number of the edge within sibling edges (`ord`).

Attribute Mapping. In this mapping an extra table for each edge name (so-called attribute) is established. The structure of these tables is similar to the previous case:

$\texttt{Edge}_{\text{name}}\texttt{(source, ord, flag, targ)}$

Universal Mapping. This method stores edges of *T* in so-called *universal table*, which contains columns for all the attribute names described in previous method. The universal table corresponds to the result of an outer join of all tables from attribute mapping. If $a_1,...,a_k$ are all the attribute names in the XML document, the universal table can have the following structure:

$\texttt{Uni(source, ord}_{a1}\texttt{, flag}_{a1}\texttt{, targ}_{a1}\texttt{,... ord}_{ak}\texttt{, flag}_{ak}\texttt{, targ}_{ak}\texttt{)}$

Obviously the universal table contains many NULL values.

Normalized Universal Mapping. This method tries to solve the main disadvantage of universal mapping storing multi-valued attributes in separate *overflow tables*. An overflow table is established for each attribute name, while its structure is the same as in attribute mapping. The universal table then contains only one row per each attribute name, others are stored in corresponding overflow tables.

Table-Based Mapping. A typical representative of the approach that enables to store only a certain collection of XML documents having similar structure is called *Table-based mapping* [2]. It is based on the assumption, that the stored XML documents have a regular structure reflecting database, tables, rows, and columns. The mapping between elements and relations is then exactly defined by the structure of the XML document.

3.2 Schema-Driven Mapping Methods

Schema-driven mapping methods are based on existing schema S_1 of stored XML documents, which is mapped to a database schema S_2. The data from XML documents valid against S_1 are then stored into relations of S_2. The purpose of these methods is to create optimal schema S_2, which consists of reasonable amount of relations and whose structure corresponds to the structure of S_1 as much as possible. All of these methods try to improve the basic mapping idea "to create one relation for each element composed of its attributes and to map element-subelement relationships using keys and foreign keys".

These methods can be further classified either according to the type of the source schema or the target schema. In the former case we usually distinguish whether the schema is specified in DTD or XML Schema, in the latter case two possibilities are considered – relational or object-relational. In both cases the methods are trying to exploit advantages of the appropriate schemes.

A more interesting classification according to the basic principles of the schema-driven approaches includes two classes – fixed and flexible methods. *Fixed methods* do not use any other information than the source schema itself; their mapping algorithm is straightforward. On the other hand, *flexible methods* are methods, which do use the additional information (usually query statistics, element statistics, etc.) and focus on creating an optimal schema for a certain application.

XMLSchemaStore Mapping. An example of fixed schema-driven mapping methods based on mapping XML Schema structures to object-relational schema is own formerly proposed mapping method called *XMLSchemaStore mapping* [6]. The method focuses on object-oriented features and integrity constraints of XML Schema language and exploitation of object-relational items of SQL standard (e.g. UDTs, references, typed tables, nesting, etc.).

The mapping algorithm is based on traversing a directed graph called *DOM graph*, whose ordered edges determine the "order" in which the UDTs and corresponding typed tables should be created to follow reference properties. The DOM graph results from the structure of a DOM tree of the given XML Schema file. It can be created the following way:

– The original edges of the DOM tree are directed to express the “direction” of element-subelement or element-attribute relationship.
– New edges expressing the “direction” of the usage of globally defined items (e.g. elements, complex types, etc.) are added.

An example of an XML Schema file and corresponding DOM graph is depicted in Figure 2. The solid lines correspond to original edges of the DOM tree; dash-and-dot lines are the additional ones.

```
<schema>
  <complexType name="T1">
    <sequence>
      <element ref="E1"/>
    </sequence>
    <attribute name="A1" type="T2"/>
  </complexType>
  <element name="E1" type="string">
  <simpleType name="T2">
    <restriction base="string">
      <length value="5"/>
    </restriction>
  </simpleType>
</schema>
```

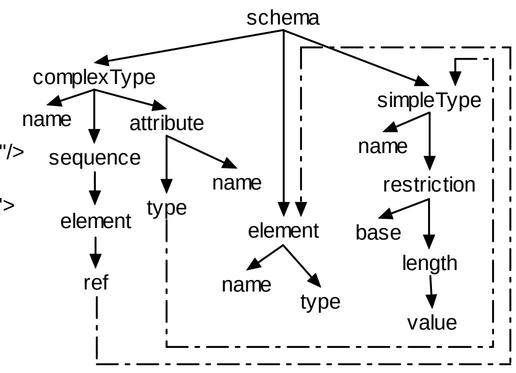

Fig. 2. An example of a DOM graph

3.3 User-Defined Mapping Methods

User-defined mapping methods are most often used in commercial systems. This approach requires that the user first defines target schema S_2 and then expresses required mapping using a system-dependent mechanism. At present, most of existing systems support some kind of user-defined mapping.

Obviously, this approach is the most flexible one. On the other hand, it requires large development effort and moreover mastering of two distinct technologies (XML and DBMS).

4 Possible Improvements

The main aim of this paper is to propose and discuss possible improvements and optimalizations of existing methods.

The first possibility is indicated by the above-mentioned flexible methods. Currently there are two most interesting and relatively different approaches – so-called *LegoDB mapping* [1] and *Hybrid object-relational mapping* [4]. The former approach is based on the idea to explore a space of possible XML-to-relational mappings and to select the best one according to given statistics including information about a sample set of XML documents and queries. The latter approach tries to improve the straightforward mapping of all elements and attributes in a DTD to relations, which can lead to large database schemes, by storing structured parts of the DTD in relations and semistructured parts in so-called *XML data type*, which supports path queries and fulltext operations on XML fragments. The main and obvious disadvantage of these methods is that the improvements are suitable only for the given application. Thus the opened problem is a solution of the situation when the set of typical queries and data changes.

Another possible improvement could focus on a combination of generic and schema-driven methods. The idea is based on exploitation of algorithms for generating XML schemes for a given set of "similar" XML documents. But since this idea brings an important question whether it is possible to generate "better" schemes automatically, it would be essential to define suitable metrics for XML schemes (such as, e.g., normal forms for relations). Moreover, such metrics would enable establishing algorithms for improving XML schemes.

Last but not least there is a problem of implementation of all axes of XPath that can be considered as a reasonable minimal requirement for XML querying. In case of schema-driven methods this would probably require storing some more information, which then will have to be searched through. Thus the most important opened issue is to what extent and how effectively it is possible to implement XML query languages using (O)RDBMS and schema-driven methods.

5 Conclusion

This paper was trying to give an overview of existing techniques for processing XML documents in (O)RDBMS and to propose and discuss their possible future optimalizations. All the mentioned ideas raise another considerable questions and opened problems. The most considerable one is whether and to what extent the exploitation of (O)RDBMS in the above-described way and its further improvements are worth the effort.

References

1. Bohannon P., Freire J., Roy P., Siméon J. From XML Schema to Relations: a Cost-Based Approach to XML Storage. Proceedings of the *XVIII. International Conference on Data Engineering ICDE 2002*. San Jose, California, 2002.
2. Bourret R. www.rpbourret.com, XML Programming, Writing, and Research.
3. Florescu D., Kossmann D. Storing and Querying XML Data Using an RDBMS. *IEEE Data Engineering Bulletin, Vol. 22, No 3*. 1999.

4. Klettke M., Meyer H. XML and Object-Relational Database Systems – Enhancing Structural Mappings Based on Statistics. Proceedings of the *III. International Workshop WebDB 2000*. Springer-Verlag, London, UK, 2001. ISBN 3-540-41826-1.
5. Megginson D. www.saxproject.org.
6. Mlynkova I., Pokorny J. From XML Schema to Object-Relational Database – an XML Schema-Driven Mapping Algorithm. Proceedings of *IADIS International Conference WWW/Internet 2004*. Madrid, Spain, October 2004. ISBN 972-99353-0-0.
7. Mlynkova I., Pokorny J. XML in the World of (Object-)Relational Database Systems. Proceedings of the *XIII. International Conference ISD 2004*. To appear in Kluwer.
8. Wood L., Apparao V., Byrne S. et al. www.w3.org/TR/REC-DOM-Level-1/, Document Object Model (Core) Level 1.