XML IN THE WORLD OF (OBJECT-)RELATIONAL DATABASE SYSTEMS

Irena Mlynkova and Jaroslav Pokorny[a]

1. INTRODUCTION

XML[1] is universally recognized as the standard for interchange and device-independent representation of information. On the other hand, XML is recently understood as a new approach to data modelling[2, 3] A well-formed XML document or a set of documents is an XML database and the associated DTD or schema specified in the language XML Schema[4] is its database schema. Implementation of a system enabling us to store and query XML documents efficiently is developed today in different ways. We do not discuss here native storage solution, i.e. a DBMS dedicated to manage XML data collections. A more practical possible solution can be found in storing XML data in (object-)relational DBMS. Moreover, this approach enables to provide XML with missing database mechanisms (e.g. indexes, transactions, multi-user access, etc.).

Currently there is a relatively large number of works devoted to storing XML data, including the special architectures like PDOM, CMS, XML Servers, XML Query Engines, etc. We refer reader to Bourret[2] for their comprehensive overview. Our contribution is a summarization of recent XML storage techniques based on today's (object-)relational database technologies, their comparing and evaluation. We also present own algorithm for mapping XML Schema structures to object-relational (OR) schema. A more comprehensive discussion can be found in Mlynkova and Pokorny[5].

For transferring the data between XML documents and (O)R structures so-called mapping methods are of a great importance. A basic classification[6] of existing mapping methods includes the 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.

Sections 2-4 contain an overview and possible classifications of the respective methods. Their evaluation and discussion is in Section 5. Section 6 provides conclusions.

[a] Charles University, Faculty of Mathematics and Physics, Department of Software Engineering, Malostranske nam. 25, 118 00 Prague 1, Czech Republic, {mlynkova,pokorny}@ksi.ms.mff.cuni.cz

2. GENERIC MAPPING METHODS

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

- a general (O)R schema into whose relations any XML document regardless its structure can be stored, or
- a special kind of (O)R 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.

2.1 Generic-Tree Mapping

A typical representative of generic mapping is a group of methods called *generic-tree mapping*[7]. An example of an XML document and its *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>
...
```

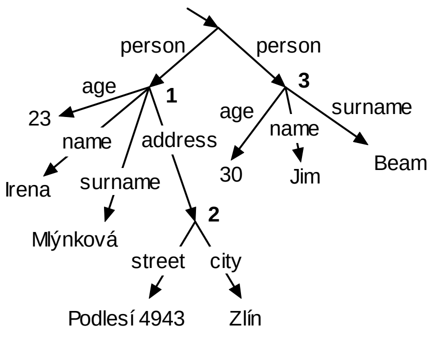


Fig. 1. An example of a generic-tree

There are several methods for storing *T*, so-called edge, attribute, universal, and normalized universal mapping.
Edge Mapping. This method stores all edges of *T* in the following table:
`Edge(source, ord, name, flag, target)`
The table contains identifiers of nodes connected by the edge (`source` and `target`), 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:
$\texttt{Edge}_{\texttt{name}}\texttt{(source, ord, flag, target)}$
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. In other words, a 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:
$\texttt{Uni(source, ord}_{\texttt{a1}}\texttt{, flag}_{\texttt{a1}}\texttt{, target}_{\texttt{a1}}\texttt{,...,ord}_{\texttt{ak}}\texttt{, flag}_{\texttt{ak}}\texttt{, target}_{\texttt{ak}}\texttt{)}$

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, so-called *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.

There is also a plenty of variations of these methods. First, in all described approaches the values in leaves can be stored either in separate *value tables* (each holds values of a certain type) or in additional columns of existing tables. Other, so-called *hybrid methods* can be created using combinations of the described approaches.

2.2 Structure-Centred Mapping

The structure-centred mapping[8] considers all nodes of the tree T having the same structure defined as a tuple $n = (t, l, c, n)$, where t is the type of the node (e.g. ELEMENT, ATTRIBUTE, TEXT,...), l is the node label, c is the node content and $n = \{n_1,...,n_n\}$ is the list of successor nodes. The paper[8] considers the problem how to realize mapping of the lists of successor nodes. It proposes three kinds of storage strategies focusing on speeding up the access performance.

Foreign Key Strategy. Each tree node n is simply mapped to a tuple with a unique identifier and a foreign key reference to the parent node. The method is quite simple and the stored tree can easily be modified. Nevertheless, its disadvantage is evident – the retrieval of the data involves many self-join operations.

DF Strategy. In this strategy each node of T is given an index value (a couple of minimum and maximum DF values), which represents its position in T. The DF values are determined when traversing T in a depth first (DF) manner. A counter is increased each time another node is visited. If a node n is visited the first time its minimum DF value n_{min} is set to the current counter value. When all child nodes have been visited, the maximum DF value n_{max} is set to the current counter value (see Fig. 2).

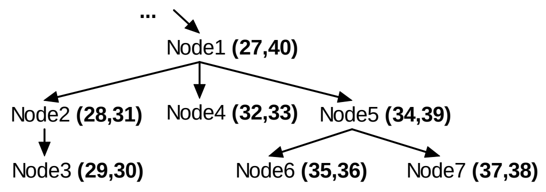


Fig. 2. An example of DF indexing

Using DF values relationships of nodes (e.g. sibling order, element-subelement relationship, etc.) can easily be determined just by comparisons. For example, a node n is a descendant of node m, if $n_{min} > m_{min}$ and $n_{max} < m_{max}$. Moreover, as the nodes can be totally ordered according to DF values, retrieving a part of a document is linear. The weak point of this strategy is document update – in the worst case it requires to update DF values of all nodes of the tree.

SICF Strategy. In this strategy each node of the graph is also given by an identification of its position – in this case so-called *simple continued fraction* (SICF)

$$s = \cfrac{1}{q_k + \cfrac{1}{\cfrac{\dots}{q_2 + \cfrac{1}{q_1}}}}$$

where $q_i \in N$ (i = 1,...,k) are called *partial quotients* of s and the expression $<q_1,...,q_k>$ *partial quotient sequence*. Sequences uniquely determine fractions and vice versa. The SICF values are determined in the following way: the root node gets a ***seed value*** $s \in N$, $s > 1$ (its SICF value is $<s>$). If a node n has SICF value $<q_1,...,q_m>$ and has n ordered child nodes $n_1,...,n_n$, then the SICF value for i-th child node is $<q_1,...,q_m, i>$.

The advantages and disadvantages of this strategy are similar to the previous one.

2.3 Simple-Path Mapping

This method[9] assumes that queries over the stored XML data are path queries of an XML query language. The main idea is to decompose XML documents into so-called *simple paths* and to store them in the database. Each simple path is based on the relation parent-descendant. Hence, each node in the graph retains its simple path. But as a simple path contains neither position nor order information, these two are stored in the graph too. The position information (called *region*) is a pair of a start and an end value, which are assigned as follows: Each word occurrence is assigned an integer number corresponding to its position within the document. Each tag is assigned a real number – its integer part indicates the position of the preceding word and its decimal part indicates the position of the tag being concerned in the current sequence of tags. The order information is composed of occurrence plus and occurrence minus order information, which expresses the index number of the node within its parent node (see Fig. 3).

All the information about *T* is stored in following four relations:

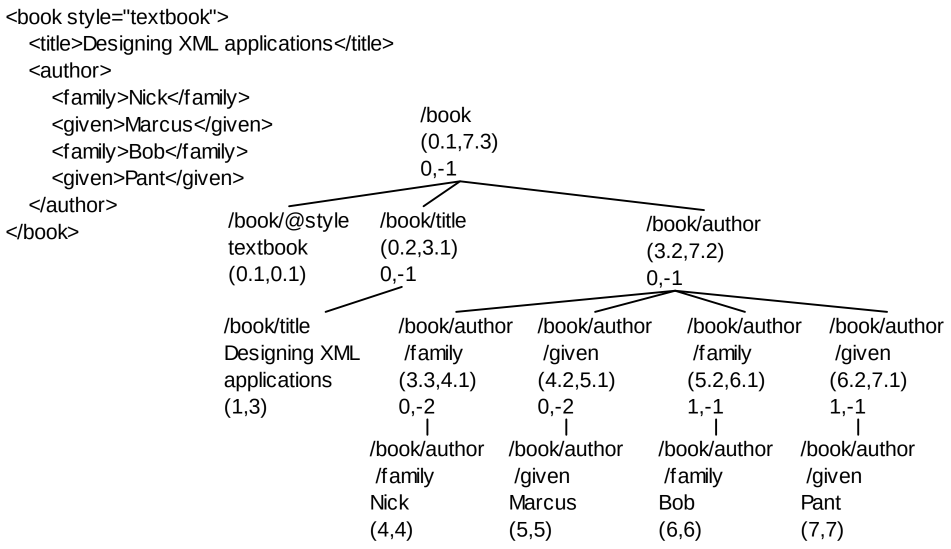
```
Element(docID, pathID, index, reindex, pos)
Attribute(docID, pathID, attvalue, pos)
Text(docID, pathID, textvalue, pos)
Path(pathexp, pathID)
```

First three relations store information about each node type – document identifiers (`docID`), path identifiers (`pathID`), plus and minus occurrence order (`index` and `reindex`), regions (`pos`), attribute and text values (`attvalue` and `textvalue`). The relation Path stores simple paths (`pathexp`) and path identifiers (`pathID`).

The main advantage of this method is apparent – storing simple paths of elements and attributes simplifies and speeds up processing path queries.

2.4 Monet Mapping

The tree model of XML data in the *Monet mapping*[10] is slightly different than in the previous methods (see Fig. 4). The main idea of this method is based on a complete binary fragmentation of *T* to binary associations, which describe different parts of the tree (edges, attributes, the topology of the document).



```
<book style="textbook">
   <title>Designing XML applications</title>
   <author>
      <family>Nick</family>
      <given>Marcus</given>
      <family>Bob</family>
      <given>Pant</given>
   </author>
</book>
```

/book (0.1,7.3) 0,-1
/book/@style textbook (0.1,0.1)
/book/title (0.2,3.1) 0,-1
/book/author (3.2,7.2) 0,-1
/book/title Designing XML applications (1,3)
/book/author /family (3.3,4.1) 0,-2
/book/author /given (4.2,5.1) 0,-2
/book/author /family (5.2,6.1) 1,-1
/book/author /given (6.2,7.1) 1,-1
/book/author /family Nick (4,4)
/book/author /given Marcus (5,5)
/book/author /family Bob (6,6)
/book/author /given Pant (7,7)

Fig. 3. An example of a simple-path tree

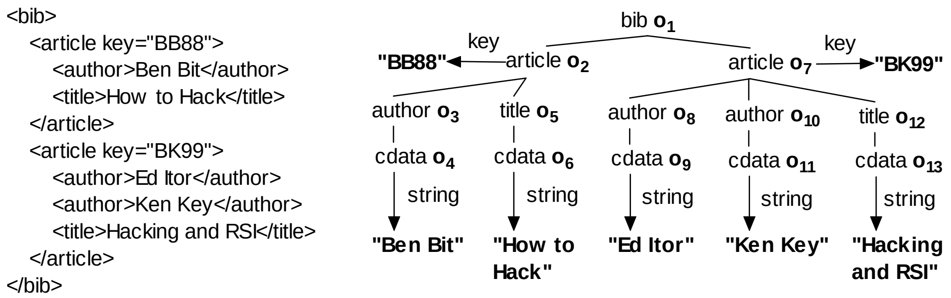
The associations, which bear semantically related information, are stored in relations together. Such information is related to definition of a path(*o*) as a sequence of (vertex and edge) labels along the path from the root node to *o* (where $\to_e$ and $\to_a$ denotes edge to an element and attribute, respectively), e.g.:

path(o_3) = bib $\to_e$ article $\to_e$ author

path("Ben Bit") = bib $\to_e$ article $\to_e$ author $\to_e$ cdata $\to_a$ string

Each path then describes the position of an element in *T* relative to the root node. At the same time, path(*o*) is used to denote the type of binary association (. , o). All associations of the same type are stored in the same binary relation.

The advantage of this method is, that it avoids large and expensive scans over irrelevant data, the disadvantage is the high degree of fragmentation, which can increase efforts to reconstruct the original document or its parts.



```
<bib>
   <article key="BB88">
      <author>Ben Bit</author>
      <title>How to Hack</title>
   </article>
   <article key="BK99">
      <author>Ed Itor</author>
      <author>Ken Key</author>
      <title>Hacking and RSI</title>
   </article>
</bib>
```

Fig. 4. An example of a Monet tree

2.5 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 exactly defined by the structure of the XML document. Apparently, this method is suitable especially for transferring the data between two relational DMBSs.

3. SCHEMA-DRIVEN MAPPING METHODS

Schema-driven mapping methods are based on existing schema S_1 of stored XML documents, written in DTD or XML Schema, which is mapped to (O)R 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".

3.1 Common Characteristics

Schema-driven mapping methods have several common basic principles[6] resulting from information stored in the XML. The most important ones are:

- Subelements with `maxOccurs = 1` are (instead of to separate tables) mapped to tables of parent elements (so-called *inlining*).
- Elements with `maxOccurs > 1` are mapped to separate tables. Element-subelement relationships are mapped using keys and foreign keys.
- Alternative subelements are mapped to separate tables (analogous to the previous case) or to one universal table (with many nullable fields).
- If it is necessary to preserve the order of sibling elements, the information is mapped to a special column.
- Elements with mixed content are usually not supported.
- A reconstruction of an element requires joining several tables.

3.2 Possible Classifications

The considered methods have several common features according to which they can be classified quite differently.
Source XML Schema. An obvious classification is based on the type of S_1. Most of these methods are based on DTD. The reason for this is, that although the DTD is quite simple, it is still sufficient for most applications. On the other hand, although the XML Schema is much more complex and thus difficult for learning, it contains useful features that DTD lacks and gives users more powerful tool for describing the allowed structure of XML documents. At present, there are also several methods (e.g. XMLSchemaStore mapping or LegoDB mapping), which try to exploit these features.

Target Database Schema. The methods differ also according to the S_2. In this paper two possibilities are concerned – relational or object-relational approach. Most of the methods are based on the former one, since the relational databases and their features managed to gain more focus than others (including OR ones). Despite of this fact there are several methods, which try to take the advantage of OR features, such as NF^2-relations (e.g. Hybrid object-relational mapping) or user defined data types and references (e.g. XMLSchemaStore mapping).
Flexibility. Another classification[6, 11] includes two classes – fixed and flexible methods. Fixed methods (e.g. Basic, Shared, and Hybrid algorithms, etc.) are those, which do not use any other information than S_1 itself and whose mapping algorithm is straightforward. On the other hand, flexible methods (e.g. LegoDB mapping or Hybrid object-relational mapping) use the additional information (e.g. query statistics, element statistics, etc.) and focus on creating an optimal schema for a certain application.

3.3 Algorithms Basic, Shared, Hybrid, and Derived Algorithms

The best-known representative of fixed schema-driven mapping methods is a group of three algorithms for mapping a DTD to relational schema called Basic, Shared, and Hybrid[12]. The main idea is based on a definition of a directed graph, so-called ***DTD graph***, which represents the processed DTD. Nodes of the graph are elements (which appear exactly once), attributes, and operators (which appear as many times as in the DTD). Edges of the graph represent element-attribute, element-subelement or element-operator and operator-subelement relationships. Each DTD is also first pre-processed and simplified to contain only ? and * operators and flat expressions (see Fig. 5).

```
<!ELEMENT author(name?,surname)>
<!ELEMENT name(#PCDATA)>
<!ELEMENT surname(#PCDATA)>
<!ELEMENT book(author*,title)>
<!ATTLIST book published CDATA>
<!ELEMENT title(#PCDATA)>
<!ELEMENT article(author)>
<!ATTLIST article paper CDATA>
```

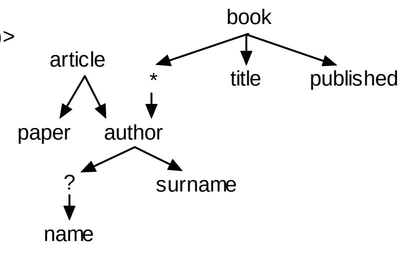


Fig. 5. An example of a DTD graph

These algorithms try to gradually improve the idea "to create one relation for each element". They differ according to the amount of redundancy they may cause.
Basic Algorithm. The Basic algorithm combines two approaches:

- to inline as many descendants of an element as possible and
- to create a relation for each element in the DTD graph.

In the former case only two kinds of element-subelement relationships are solved using keys and foreign keys – subelements with multiple occurrence (indicated by the use of * operator) and recursion (indicated by cycles in the graph). The main disadvantages of this algorithm are obvious – a huge amount of unnecessary relations and a great deal of redundancy since an element node can be represented in several relations.

Shared Algorithm. The Shared algorithm tries to avoid the drawbacks of Basic. The idea is to identify elements that are represented in multiple relations and to share them by creating separate relations for them. The mapping rules are:

- Nodes with an in-degree of one are inlined to parent relations.
- Nodes with an in-degree of zero are stored in separate relations.
- Repeated elements are stored in separate relations.
- Of all mutually recursive elements having an in-degree one, one of them is stored in a separate relation.
- The problem of inlined elements, which can become roots of an instance XML document, is solved using a flag for each element that indicates this state.

Apparently the main advantage of the Shared algorithm is the reduced amount of relations and redundancy. Its main disadvantage is the number of join operations necessary for restoring an element, which can be worse than in Basic.

Hybrid Algorithm. The Hybrid algorithm tries to combine the join reduction properties of Basic with the sharing features of Shared. The algorithm is similar to Shared except for additional inlining of elements with an in-degree greater than one, that are neither recursive nor reached through a * node.

CPI Algorithm. CPI (Constraints-Preserving Inlining) method[13] can be based e.g. on the mentioned Hybrid algorithm. Its main purpose is to capture not only the structure of the DTD but the semantic constraints as well. The considered constraints are e.g. domain constraints, cardinality constraints (i.e. +, *, ? operators), referential integrity (i.e. ID, IDREF, IDREFS types), etc. These constraints are represented using corresponding SQL constraints e.g. NOT NULL, UNIQUE, PRIMARY/FOREIGN KEY, CHECK, etc.

3.4 Object-Relational Mapping

Object-relational mapping[14] uses the word "object-relational" in a bit confusing way, since it does not denote the type of S_2 but the two steps of the algorithm. S_1 is expressed either in DTD or XML Schema; S_2 is relational in all cases. The two steps are:

1. S_1 is mapped to an object schema expressed in an object-oriented language.
2. The object schema is mapped to S_2.

Obviously, if the object schema is not essential, it can be eliminated.

The object schema models S_1 as a tree of objects. In this step element types with PCDATA-only content and attribute types are considered as ***simple types***. Element types with element or mixed content, or element types with attributes are considered as *complex types.* The mapping rules can be summed as follows:

- simple types $\rightarrow$ scalar data types,
- complex types $\rightarrow$ classes with each element type in the content model mapped to a property of the class – the data type of each property is either the scalar data type or a pointer/reference to the corresponding object,
- attributes $\rightarrow$ properties,
- subelements in a sequence or a choice $\rightarrow$ properties (whereas in the latter case the corresponding columns in the relational schema will be nullable),
- repeated subelements $\rightarrow$ multi-valued properties of (un)known size,

- mixed content → a multi-valued property for storing PCDATA-values plus additional *order columns* for each property sharing the same order space.

An example of a DTD and associated schemes is in Fig. 6.

DTD:	object schema:	relational schema:
<!ELEMENT A(B,C)>	class A {	A(b,c_{fk})
<!ELEMENT B(#PCDATA)>	String b;	
	C c; }	
<!ELEMENT C(D,E)>	class C {	C($\underline{pk}$,d,e,f)
<!ELEMENT D(#PCDATA)>	String d;	
<!ELEMENT E(#PCDATA)>	String e;	
<!ATTLIST E F CDATA>	String f; }	

Fig. 6. An example of object-relational mapping for DTD

For XML Schema the transformation is similar, the differences are related to additional features XML Schema has. The step doing object-to-relational transformation does not distinguish from usual approaches used in today's software engineering.

3.5 Constraints Preserving Mapping

Constraints preserving mapping[15] preserves not only the structure of S_1 but also the variety of semantic constraints XML Schema enables to express.

XML Schema structures are formally represented by the regular tree grammar called FD-XML[15]. An extension of ER model, so-called EER model, is proposed and the FD-XML is converted into EER schema. Then, the EER schema is simplified and optimized, preserving both the structure and the semantic constraints and finally, the simplified EER schema is converted to relational schema. The EER model uses (min, max) cardinalities, arrowheads modelling parent-child relationships, and accessories in order to preserve the data constraints (see Fig. 7).

```
<schema>
  <element name="E1" type="T1"/>
  <complexType name="T1">
    <sequence>
      <element name="E2" type="string"/>
      <element name="E3" type="int"/>
      <element name="E4" type="decimal  minOccurs="0"/>
    </sequence>
    <attribute name="A1" type="string"/>
  </complexType>
</schema>
```

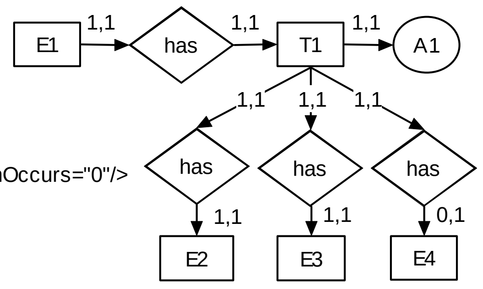


Fig. 7. An example of mapping an XML schema to an EER diagram

The rules for simplification of the EER schema include converting an entity to its parent entity's attribute and removing a subentity from its parent entity if possible. The rules for transforming the simplified EER schema to relational schema are similar to well-known algorithms for design of relational databases.

3.6 XMLSchemaStore Mapping

XMLSchemaStore mapping[5] maps S_1 expressed in XML Schema to OR schema expressed in SQL:1999[b] standard[16]. It tries to preserve the structure as well as semantic constraints of the S_1 in the S_2 and to exploit OR features of the SQL:1999 standard.

The mapping rules are as follows:

- built-in and user-defined simple type $\rightarrow$ corresponding database simple type (eventually) together with corresponding integrity constraint(s),
- complex type and model group $\rightarrow$ OR user-defined type (UDT), whereas:
  - o XML attributes $\rightarrow$ UDT attributes with corresponding simple types,
  - o simple element content $\rightarrow$ UDT attribute with corresponding simple type,
  - o element-subelement relationship $\rightarrow$ UDT attribute, whose type is (according to the allowed occurrence and the type of the subelement) either the UDT of the subelement or the REF/ARRAY of REF to the UDT,
- deriving of complex types $\rightarrow$ UDT inheritance,
- element (according to its type and allowed occurrence) $\rightarrow$ own typed table[c] or a typed column of the table which corresponds to its parent element.

S_2 can be then described as a set of typed tables connected using references.

The mapping algorithm is based on traversing a directed graph called *DOM graph* (see Fig. 8), whose edges determine the "order" in which the UDTs and typed tables should be created to follow reference properties.

```
<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>
```

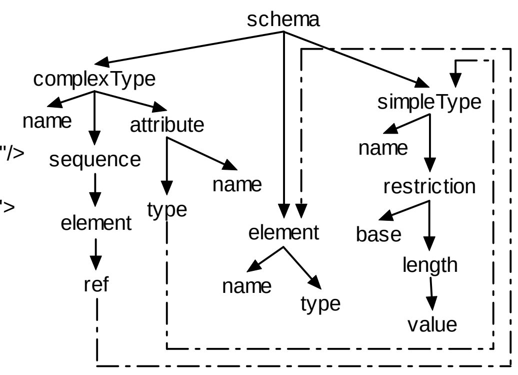


Fig. 8. An example of a DOM graph – the solid lines correspond to original edges of the DOM tree; dash-and-dot lines are the additional ones

The DOM graph results from the structure of a DOM tree of the given XML Schema file in the following way:

[b] Latterly the SQL:2003 standard is at disposal. Its new type MULTISET can be used for unordered XML data.

[c] A table that is defined based on a UDT, i.e. rows of a typed table are instances of the corresponding UDT.

- 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.

The mapping is done while traversing the graph starting in `schema` node. First, all descendants of a current node are processed, e.g. the UDTs and typed tables are created. Second, the current node can be processed, since all necessary OR items already exist.

3.7 LegoDB Mapping

A representative of flexible schema-driven mapping methods is an algorithm proposed in LegoDB system[11]. First the method defines fixed mapping of XML Schema structures (for processing simplicity rewritten into syntactically simpler, but semantically equivalent *p-schemas*) to relations. The flexibility 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.

In order to select the best mapping the system in turns applies the following two steps to the source p-schema, until a good result is achieved:

1. Any possible XML-to-XML transformation is applied to the p-schema.
2. XML-to-relational transformations are applied to the new p-schema and against the resulting relational schema the given queries are estimated.

As the space of possible p-schemas can be large (possibly infinite), the paper[11] also proposes a greedy evaluation strategy that explores only the most interesting subset.

The XML-to-XML transformations used in the algorithm are: *inlining*/*outlining*, *union factorization*/*distribution*, *repetition merge*/*split*, *wildcards rewriting*, and *from union to options*. The XML-to-relational transformations are similar to those described in the previously mentioned methods.

3.8 Hybrid Object-Relational Mapping

Another example of flexible schema-driven mapping methods is a hybrid object-relational mapping[17]. It 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 types*, which support path queries and fulltext operations for XML fragments.

The main concern of this approach is to decide which parts of the DTD are structured and which semistructured. The suggested algorithm is as follows:

1. A graph (similar to above-described DTD graph) is built.
2. A *measure of significance* w is determined for each element/attribute.
3. The resulting database design is derived from the graph.

The measure w can be expressed as

$$w = \frac{1}{2}w_S + \frac{1}{4}w_D + \frac{1}{4}w_Q$$

where the used variables (weights) w_S, w_D, and w_Q are derived from the DTD structure, the existing XML data, and the queries, respectively.

According to a given limit of w (which influences the level of detail of S_2) the algorithm determines non-leaf nodes n, each of which fulfils the following conditions:

1. All descendants of n are below the given limit.
2. There exists no predecessor of n that fulfils the condition 1.

All subgraphs consisting of these nodes and their descendants are replaced by an XML attribute. The resulting graph is finally mapped using a fixed mapping method to OR schema. The mapping (see Fig. 9) focuses on the use of structured or nested attributes in NF^2-relations, assuming the existence of SET[d] and TUPLE constructors.

```
<!ELEMENT chapter(ctitle, section+)>     chapter=<id,ctitle,{section}>
<!ATTLIST chapter id ID REQUIRED>
<!ELEMENT ctitle(#PCDATA)>

<!ELEMENT section(stitle, paragraph+)>   section=<id,stitle,{paragraph}>
<!ATTLIST section id ID REQUIRED>
<!ELEMENT stitle(#PCDATA)>
                                         chapter = <id,ctitle,{<id,stitle,{paragraph}>}>
```

Fig. 9. An example of hybrid object-relational mapping

4. USER-DEFINED MAPPING METHODS

User-defined mapping methods are most often used in commercial systems. This approach requires that the user first defines S_2 and then expresses required mapping using a system-dependent mechanism, e.g. a special query language, a declarative interface, etc. 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 relational DBMS)[11]. The description of these methods exceeds the scope of this paper.

5. DISCUSSION OF MAPPING METHODS

Usually XML documents are classified into two groups according to their content, structure, and supposed use – ***data-centric*** and ***document-centric***[2]. The structure of data-centric documents is typically known and is specified in DTD or XML Schema. In document-centric documents the structure is typically specified using “mixed-content models” with arbitrary inter-leaving of text with XML mark-up. The distinction between these two groups is not generally obvious (documents which belong to both groups are called ***hybrid*** documents). A usability of mapping methods depends on these categories.

Notice, there is probably no reasonable argument for comparing generic and schema-driven methods together. Table 1 summarizes all discussed features of generic methods. Generally speaking, these methods are most suitable in cases when no XML schema exists. All mentioned methods are primarily determined for data-centric XML

[d] Compare with ARRAY in SQL:1999 or MULTISET in SQL:2003.

documents, but probably with extensions related to the above-mentioned document-centric items they could be used for document-centric documents as well.

Table 1. Summary of generic mapping features

Mapping	Considered features			
	CDATA sections, comments,...	Mixed content elements	Different data types	Preserving the element order
Generic-tree	Not supported	Not supported	Supported	Sibling order
Structure-centred	Not supported	Supported	Not supported	Total ordering
Simple-path	Not supported	Supported	Not supported	Sibling order
Monet	Not supported	Supported	Not supported	Sibling order

All mentioned features of schema-driven methods are summarized in Table 2. The idea of flexible mapping methods is relatively new. There is no reason for asking whether flexible methods are better than the fixed ones – apparently they are since the resulting schema suits the given statistics at least as well as corresponding fixed method. Indeed these methods can be obviously used only if it is possible to obtain necessary statistic information. The interesting point is how to determine the “best” schema. Just two but nevertheless quite different representatives were mentioned, whereas both are somehow based on a sample set of XML documents and typical queries. Obviously, the most flexible mapping is provided by user-defined mapping methods. Last but not least there is the matter of the S_2 schema. Without any doubts an OR schema solves the problems of multi-valued properties in more natural way than it does a relational schema in the 1NF.

Table 2. Summary of schema-driven mapping features

Mapping	Considered features				
	Source schema	Target schema	Mixed content elements	Preserving sibling order	Flexibility
Basic, Shared, Hybrid, CPI	DTD	Relational	Not considered	Can be extended	Fixed
Object-relational	DTD/XML Schema	Relational	Supported using additional fields	Supported	Fixed
Constraints preserving	XML Schema	Relational	Not considered	Not considered	Fixed
XMLSchema-Store	XML Schema	Object-relational	Not supported	Supported	Fixed
LegoDB	XML Schema	Relational	Not considered	Not considered	Flexible
Hybrid object-relational	DTD	Object-relational	Supported using XML-aware type	Can be extended	Flexible

To sum up, schema-driven mapping methods try to exploit the information in the given XML schema as much as possible. Although they can preserve some document-centric features (e.g. document order or mixed content elements), they are usually used for data-centric XML documents.

6. CONCLUSIONS

This paper was trying to offer a general and clear summary of existing strategies for connecting XML and database technologies, especially those related to relational and OR systems. Several possible classifications were mentioned and discussed and the best-known representatives of the classes were briefly described. Finally the general common features, advantages, and disadvantages of the described methods were discussed.

There are several areas, which will probably be in the main focus of future works. The first will apparently concern semantic constraints (especially those expressed using XML Schema) that should be preserved in the target (O)R schemes. Several of the mentioned methods partly focused on this area, but the current features of DBMSs and relational languages still limit these approaches in many ways.

The second interesting point is connected with flexible mapping methods, which try to optimize the fixed schema according to its future use. As there are no rules, which define a “good” XML schema (such as, e.g., normal forms for relations), the fixed mapping of a “bad” one can result in a “bad” relational schema. Thus an important task may be to determine a definition of a “good” XML schema and ways how to establish it.

ACKNOWLEDGMENTS

This research was supported in part by GACR grant 201/02/1553.

REFERENCES

1. Extensible Markup Language (XML) 1.0 (Third Edition) W3C Recommendation (February 4, 2004); www.w3.org/TR/REC-xml/.
2. XML and Databases (2003); www.rpbourret.com.
3. J. Pokorny, XML: a challenge for databases?, in *Contemporary Trends in Systems Development*, edited by Maung K. Sein (Kluwer Academic Publishers, Boston, 2001), pp. 147 – 164.
4. XML Schema Part 0: Primer W3C Recommendation (May 2, 2001); www.w3.org/TR/xmlschema-0/.
5. I. Mlynkova and J. Pokorny, XML in the World of (Object-) Relational Database Systems, Technical Report No. 2003-8, Dep. of Software Engineering, Charles University, 2003, 28 p.
6. S. Amer-Yahia and M. Fernandez, Overview of Existing XML Storage Techniques, AT&T Labs, 2001.
7. D. Florescu and D. Kossmann, Storing and querying XML data using an RDBMS, *IEEE Data Engineering Bulletin*, Vol. 22(3), pp. 27 – 34 (1999).
8. A. Kuckelberg and R. Krieger, Efficient structure oriented storage of XML documents using ORDBMS, *Springer-Verlag Heidelberg*, Vol. 2590, pp. 131 – 143 (2003).
9. T. Shimura, M. Yoshikawa, and S. Uemura, Storage and retrieval of XML documents using object-relational databases, *Proc. of DESA Conf.*, pp. 206 – 217 (1999).
10. A. Schmidt, M. Kersten, M. Windhouwer, and F. Waas, Efficient relational storage and retrieval of XML documents, *Proc. of WebDB Conf.*, pp. 47 – 52 (2000).
11. P. Bohannon, J. Freire, P. Roy, and J. Siméon, From XML schema to relations: a cost-based approach to XML storage, *Proc. of ICDE Conf.*, p. 64 (2002).
12. J. Shanmugasundaram, K. Tufte et al., Relational databases for querying XML documents: limitations and opportunities, *Proc. of VLDB Conf.*, pp. 302 – 314 (1999).
13. D. Lee and W. W. Chu, CPI: constraints-preserving inlining algorithm for mapping XML DTD to relational schema, *Journal of Data & Knowledge Engineering*, Vol. 39(1), pp. 3 – 25 (2001).
14. R. Bourret, C. Bornhövd, and A. P. Buchmann: A generic load/extract utility for data transfer between XML documents and relational databases, *Proc. of WECWIS Conf.*, p. 134 (2000).
15. H. Sun, S. Zhang, J. Zhou, and J. Wang: Constraints-preserving mapping algorithm from XML-schema to relational schema, *Springer-Verlag Heidelberg*, Vol. 2480, pp. 193 – 207 (2002).
16. J. Melton, *Advanced SQL: 1999 - Understanding Object-Relational and Other Advanced Features* (2003, Morgan Kaufmann Publishers).
17. M. Klettke and H. Meyer, XML and object-relational database systems – enhancing structural mappings based on statistics, *Informal Proc. of WebDB Workshop*, pp. 151 – 170 (2000).