Managing XML documents in object-relational databases

Meike Klettke Holger Meyer

Database Research Group
Computer Science Department, University of Rostock
18051 Rostock, Germany
Email: {meike,hme}@informatik.uni-rostock.de

Abstract

XML becomes the standard for the representation of structured and semi-structured data on the Web. Relational and object-relational database systems are a well understood technique for managing and querying such large sets of structured data.

In our approach, the nested-relations data model is the basic model for representing XML data in object-relational database systems. Using the partitioned normal form (PNF), we show how a relevant subset of XML documents and their implied structure can be mapped onto database structures.

Besides straight-forward mappings, there are some XML structures which cannot be easily mapped to database structure, namely mixed-content and alternatives. These structures would result in large database schemas and sparsely populated databases. As a consequence, such XML document fragments should be mapped onto database attributes of type XML and kept as is.

We present an algorithm which finds some kind of optimal mapping based on statistics and the XML Document Type Definition (DTD). The statistics are derived from sample XML document sets and some knowledge about queries on XML document collections.

Keywords: XML, DTD, mapping, object-relational databases, partitioned normal form (PNF)

1 Motivation

At present, a lot of applications on the Web use or intend to use XML as an intermediate format for representing large amounts of structured or semi-structured data. XML [Con98] can be seen as a popular subset of the SGML markup language customized for the Web age. It is better suited for automatic processing and not as complex as SGML. If necessary, there are techniques for transforming SGML to XML [Cla97].

XML documents are said to have a well formed structure if they meet some syntactical requirements, e.g. if they have a hierarchically nested element structure. DTD are used to describe the structure of several, similarly structured XML documents. Documents satisfying such a document type definition are called valid XML documents.

Alternatively the structure of an XML document can be extrapolated from the individual document. Using only the implied structures leads to several problems. Discovering schema similarities between several XML documents, finding an optimal common schema, and handling schema evolution are only three reasons for not solely relying on valid XML documents. So we assume the existence of such a document description in the sequel.

An application field of XML, where a lot subsequent XML standards are defined, is the management of multi-media documents in large scale, distributed digital libraries.

Even if the typical digital library manages documents and information describing and supplementing these documents and resulting in similar DTDs, the contents and the usage of these document collections could vary a lot. To illustrate this, consider the following examples of popular digital libraries, even if they don't use XML:

- DBLP [1], Michael Ley, maintains about 150,000 XML documents resembling BibTeX-style references that describe his bibliographic network of database and logical programming publications, conferences, research groups, projects, and authors.

- Springer LINK-Service[2] offers fulltexts of journal publications, e.g. as articles of the VLDB journal on a subscription basis.

- Amazon Bookstore[3], a big e-commerce Web site selling books.

A possible simplified DTD for the examples above would look like our DTD in Figure 1. But the DTD is only one aspect influencing the mapping of XML structures onto databases.

Other important aspects are the frequency of element and attribute occurrences in document collections and the most often queried elements and attributes.

For example, the DBLP system contains documents in which nearly all elements occur with the same frequency. Users mainly ask for author names, conference, journal, or article titles, and browse through references sections of articles or table of contents of conference proceedings. Since DBLP stores no fulltexts, there are no chapter, sections, paragraph etc., except for the extracted references.

In contrast, articles of the VLDB journal at Springer contain many sections, paragraphs, and ordinary text `#PCDATA` in XML jargon. Nevertheless, querying these articles is based on author names, titles, volumes, or journal numbers.

At Amazon we not only have information on books, but on movies, DVD's, etc. The search is based on book titles, keywords, some kind of "what's related" classification, author names, or small ISBNs. You can additionally browse through reviews and annotations by other costumers or journalists.

All examples show how different the document collections are on an instance level and how wide the range of query profiles is. This article outlines how this information can improve the mapping of XML onto object-relational structures. But we start with a seemingly straight forward transformation of XML structures onto object-relational ones.

2 XML and object-relational databases

Object-relational databases allow the use of structured or nested attributes in NF^2-relations. These characteristics can be used to map XML documents onto databases in a more natural

[1] `http://dblp.uni-trier.de/`
[2] `http://link.springer.de/`
[3] `http://www.amazon.com/`

```
<!ELEMENT publications (book | article | conference)*>
<!-- book -->
<!ELEMENT book (front, body, references)>
<!ELEMENT front (title, author+, edition, publisher)>
<!ELEMENT title (#PCDATA)>
<!ELEMENT author (first, second, e-mail?)>
<!ELEMENT first (#PCDATA)>
<!ELEMENT second (#PCDATA)>
<!ELEMENT e-mail (#PCDATA)>
<!ELEMENT edition (#PCDATA)>
<!ELEMENT publisher (#PCDATA)>
<!ELEMENT body (part+ | chapter+)>
<!ELEMENT part (ptitle, chapter+)>
<!ATTLIST part id ID #REQUIRED>
<!ELEMENT ptitle (#PCDATA)>
<!ELEMENT chapter (ctitle, section+)>
<!ATTLIST chapter id ID #REQUIRED>
<!ELEMENT ctitle (#PCDATA)>
<!ELEMENT section (stitle, paragraph+)>
<!ATTLIST section id ID #REQUIRED>
<!ELEMENT stitle (#PCDATA)>
<!ELEMENT paragraph (#PCDATA)>
<!ELEMENT references (publications*)>
<!ATTLIST references reftype (book | article | conferences | wwwaddress)
   "article">
<!-- article -->
<!ELEMENT article (meta, body, references)>
<!ELEMENT meta (author+, title, conference)>
<!-- conference -->
<!ELEMENT conference (editor+, conftitle, city, year)>
<!ELEMENT editor (first, second, e-mail?)>
<!ELEMENT conftitle (#PCDATA)>
<!ELEMENT city (#PCDATA)>
<!ELEMENT year (#PCDATA)>
```

Figure 1: Book DTD

way. The element hierarchy of a DTD can be directly mapped onto nested attributes of a database. Furthermore, we have the possibility to use set-valued attributes to represent elements with the ’*’ and ’+’ quantifiers.

Since there are quite different object-relational models, we use a very simple data model that can be mapped onto existing systems like IBM DB2 UDB or Informix Universal Database. In addition to the set of basic data types, like `string`, `numeric`, etc. we assume only two type constructors: `set-of`, `tuple-of` only, which leads to an NF^2 database model. Moreover, we restrict this model to a subset, the partitioned normal form (PNF). In situations where element ordering matters a third type constructor `list-of` is exploited.

Regarding the basic data types, we assume the existence of an XML data type allowing the evaluation of path expressions over XML fragments besides fulltext operations on the embedded `#PCDATA` [Por99].

Relations in partitioned normal form (PNF), introduced in [RKS88], are an important subclass of nested relations. The atomic or elementary attributes of a relation in PNF are a superkey of the relation. Any non-atomic or composed attribute of the relation has to be in PNF too. A

Table 1: PNF relation as nested table

conference

conftitle	**editors**			**city**	**year**
	first	**second**	**e-mail**		
VLDB ’99	Malcom	Atkinson	. . .	. . .	’99
	Maria E.	Orlowska	. . .		
	Patrick	Valduriez	. . .		
	Stan	Zdonik	`sbz@cs.brown.edu`		
	Michael	Brodie	. . .		
ADL ’98	. . .	. . .	. . .	. . .	’98
	. . .				

flat key on each nesting level is required for relations to be in PNF.

We use the following notations: $\langle\rangle$ describes a tuple constructor, $\{\}$ represents a set constructor, and the **key** of a nested relation is in bold face.

The `conference` and `editor` elements of the DTD example would be represented in PNF as follows.

$$\begin{aligned} \text{conference} &= \langle \mathbf{conftitle}, \{\text{editor}\}, \text{city}, \text{year} \rangle \\ \text{editor} &= \langle \mathbf{first}, \mathbf{second}, \text{e} - \text{mail} \rangle \end{aligned}$$

could be combined into

$$\text{conference} = \langle \mathbf{conftitle}, \{\langle \mathbf{first}, \mathbf{second}, \text{e} - \text{mail} \rangle\}, \text{city}, \text{year} \rangle$$

A more natural representation is a nested table structure (Table 1). We use both to illustrate our transformations in this article.

2.1 Straight-forward mappings

In the following sections we provide mapping rules for several DTD structures that can easily be transformed.

2.1.1 Simple elements

Though simple elements and attributes can be used interchangeably to model several real-world aspects, there are some differences. Attributes of an element are unordered but the ordering of elements matters most times. Attributes usually have atomic values but elements can be repeated and have a complex structure, i.e. can contain other elements.

Nevertheless, simple elements just containing `#PCDATA`, or `EMPTY` elements without attributes can be mapped directly into database attributes.

2.1.2 Sequences of elements

As with elements containing attributes, sequences of elements are translated into a `tuple-of` database attributes representing the components of the sequence.

2.1.3 Optional elements `'?'`

XML elements that are optional result in database attributes that can have `NULL` values.

```
<!ELEMENT author (first, second, e-mail?)>
<!ELEMENT first (#PCDATA)>
<!ELEMENT second (#PCDATA)>
<!ELEMENT e-mail (#PCDATA)>
```

is translated into

$$\mathrm{author} = \langle \mathbf{first}, \mathbf{second}, \mathrm{e} - \mathrm{mail} \rangle$$

2.1.4 Attributes

Elements with attributes are mapped into a database attribute of type `tuple-of`. The attributes become components/database attributes of that tuple.

```
<!ELEMENT part (ptitle, chapter+)>
<!ATTLIST part id ID REQUIRED>
```

can be mapped onto

$$\mathrm{part} = \langle \mathbf{id}, \mathrm{ptitle}, \{\mathrm{chapter}\} \rangle$$

If XML attributes are `#REQUIRED`, `#IMPLIED`, have *default values*, or are *name groups* (a finite set of values), they can be transformed into corresponding database concepts, e.g. concepts of SQL-99 like `NOT NULL`, `DEFAULT VALUES`, `CREATE DOMAIN`, `CHECK`.

```
<!ATTLIST references reftype (book | article | conferences |
  wwwaddress) "article">
```

is equivalent to

```
create domain reftype varchar(12) default 'article'
  check (
    value in ('book', 'article', 'conference', 'wwwaddress'))
```

2.1.5 Set-valued attributes

However, some XML attributes not only contain atomic values but are set-valued, such as `IDREFS` and `NMTOKENS`. These attributes are transformed into a `set-of`, set-valued attribute.

2.1.6 Missing identifying elements or attributes

If there is no element or attribute with a key property (e.g. attribute of type `ID`), we add artificial keys for the PNF and the pure relational mapping.

2.1.7 Repeatable elements

XML elements which can occur zero, one, or several times are marked with `'*'`, elements which occur one or several times are marked with `'+'`. Both elements are transformed into a set-valued database attribute, where `'*'` elements are allowed to have `NULL` values, but `'+'` are not (`NOT NULL` constraint in SQL).

2.1.8 Complex combinations of elements

Taking the example definition of the `chapter` element, we can perform the following more complex transformations by combining the simple ones.

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

The attribute `id` is translated into a database attribute. The same is done with the element `ctitle`, which is of type `#PCDATA`. The element `section` which occurs one or more times within a `chapter` element is transformed into a set-valued database attribute.

$$\text{chapter} = \langle \mathbf{id}, \text{ctitle}, \{\text{section}\} \rangle$$

A similar translation can be done for the `section` element:

```
<!ELEMENT section (stitle, paragraph+)>
<!ATTLIST section id ID REQUIRED>
<!ELEMENT stitle (#PCDATA)>
```

This DTD fragment would result in:

$$\text{section} = \langle \mathbf{id}, \text{stitle}, \{\text{paragraph}\} \rangle$$

Both translations for the `chapter` and the `section` element can be combined into one definition:

$$\text{chapter} = \langle \mathbf{id}, \text{ctitle}, \{\langle \mathbf{id}, \text{stitle}, \{\text{paragraph}\} \rangle\} \rangle$$

2.2 Mapping Problems

The following XML structures are especially difficult to express in structured databases, because there are often no obvious transformations. If no further decomposition of an element is possible, we use a database attribute of type XML to store the element with its whole structure as an XML fragment.

2.2.1 Alternatives

There is no easy translation of alternatives, but we can identify some basic cases. If all elements are simple elements (of type `#PCDATA`, `EMPTY`), the mapping could be done by an attribute with the element names as values, or a `tuple-of` the element name and the corresponding `#PCDATA`. It's almost the same as the mapping of *name groups*.

If the elements of the alternative have a more complex structure, we can either store each element in one separate relation or combine the alternative into one attribute of type XML. The decision should be reached based on heuristics, which will be discussed later. Whether to divide or to combine should at least take the ratio of the structural depths of each branch, and the usage in documents and queries into account.

For instance, the following alternative on a very high structural level is a candidate for dividing into separate PNF relations.

```
<!ELEMENT publications (book | article | conference)*>
<!ELEMENT book (front, body, references)>
<!ELEMENT article (meta, body, references)>
<!ELEMENT conference (conftitle, editor+, city, year)>
```

is translated into

$$\begin{aligned}\text{publication} &= \text{book} \cup \text{article} \cup \text{conference}\\ \text{book} &= \{\langle \mathbf{front}, \text{body}, \text{references}\rangle\}\\ \text{article} &= \{\langle \mathbf{meta}, \text{body}, \text{references}\rangle\}\\ \text{conference} &= \{\langle \mathbf{conftitle}, \{\text{editor}\}, \text{city}, \text{year}\rangle\}\end{aligned}$$

2.2.2 Recursive structures

There is no recursion on the instance level but it may exists in the document type definition. Each element definition that references an element of which it is a sub-element should be marked. Elements marked are not combined into a `tuple-of` but are mapped onto a separate PNF relation.

```
<!ELEMENT publications (book | article | conference)*>
<!-- book -->
<!ELEMENT book (front, body, references)>
 ...
<!ELEMENT references (publications*)>
```

Another problem related somewhat to recursion is common sub-expression (CSE). In the following example `chapter` can be a direct member of `body` or sub-element of `part`. Such elements are marked during the DTD analysis and duplicated afterwards.

```
<!ELEMENT body (part+ | chapter+)>
<!ELEMENT part (ptitle, chapter+)>
<!ATTLIST part id ID REQUIRED>
<!ELEMENT ptitle (#PCDATA)>
<!ELEMENT chapter (ctitle, section+)>
<!ATTLIST chapter id ID REQUIRED>
<!ELEMENT ctitle (#PCDATA)>
```

2.2.3 Mixed-content type model

The so-called mixed-content type model is used to allow text spanning several elements, to mix fulltext with element structure, or to embed element structures within a fulltext. As up to now we map such elements to a single attribute of type XML. A more structured solution should take advantage of a `list-of` type constructor and allow separately querying the concatenated `#PCDATA` or the embedded elements.

2.2.4 Hyperlinks

Within the scope of XML, there are at least three different standards concerning hyperlinks: the basic `ID`, `IDREF` mechanism and the complementary XLink and XPointer standards.

The first type of hyperlinks (`ID`, `IDREF`) is part of the XML standard. These links can only be used within a single, logical document. A simple ad-hoc solution would be a translation into referential integrity constraints or the exploitation of the SQL-99 reference mechanism.

2.2.5 Entity references

This kind of indirect containment is similar to the macro mechanism in programming languages. The question arises, when to perform the macro expansion; when the document is inserted into the database or when it is extracted? For now, we store the unresolved entity reference.

2.2.6 Document ordering

In applications where the document ordering matters, we could not simply apply our mapping algorithm. We have to extend it using a type constructors for ordered sets, e.g. `list-of` for sorted lists. Alternatively, we can introduce redundancies by storing the whole XML document as it is in addition to the structured part.

3 XML and relational databases

Up to this point, we enumerated the mapping of a DTD onto an object-relational database. Even if object-relational databases are powerful tools for handling XML data, it is desirable to have the same functionality with existing relational systems.

Our translation using PNF can be easily mapped to relations in 1NF. [Hul90] presents a bijective transformation with special nest and unnest operations without information loss. Applying the unnest operation to the result of our transformation would decompose the PNF relations into 1NF relations.

Nested relations are divided into separate relations in the following way: the key of the nesting level becomes the primary key of the 1NF relation, and the key of the upper or surrounding relation becomes a foreign key of this relation.

The above example of the composed attribute `conference` results in two 1NF relations `conferences` and `editors` assuming `conftitle` is the primary key for `conferences`:

conferences

conftitle	city	year
VLDB ’99	Edinburgh	1999
ADL ’98	Santa Barbara	1998
. . .		

editors

conftitle	first	second	e-mail
VLDB ’99	Malcom	Atkinson	. . .
VLDB ’99	Maria E.	Orlowska	. . .
VLDB ’99	Patrick	Valduriez	. . .
VLDB ’99	Stan	Zdonik	`sbz@cs.brown.edu`
VLDB ’99	Michael	Brodie	. . .
. . .			

4 XML in a “hybrid” databases

In the last two sections we demonstrated a method to translate XML documents into object-relational or relational databases. The advantages of the approach are obvious; we can use databases to store and administer semi-structured data. In particular, we can use the extended query functionalities of database query languages (SQL). This approach works for applications using XML to represent structured data.

But this method doesn’t work well with the characteristics of semi-structured data. It results in very large database schemas and the resulting databases are “much less” populated. Therefore, the complete translation of XML documents into databases is not an efficient way to handle semi-structured documents.

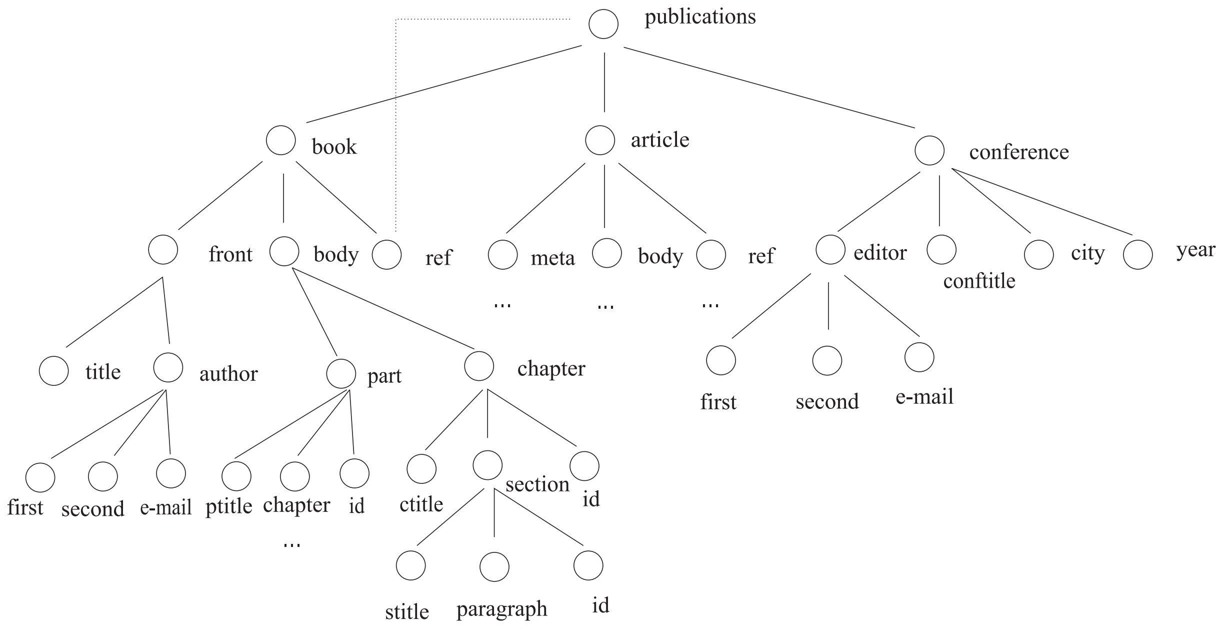


Figure 2: DTD graph for the sample DTD

One solution is to use “hybrid” databases or object-relational databases with data type XML. We want to translate some elements/attributes into attributes of a relational or object-relational database and others shall be combined as attributes with type XML.

4.1 Advantages

Using hybrid databases has two main advantages:

1. We can use database functionality for the most important and often queried terms.

2. In many applications formal and informal information belong closely together.

For example, in libraries structured data (e.g. title, author, publisher, year, etc.) exist in addition to the documents. Databases of service hotlines are another example where the structured information about devices, numbers, and so on belong closely together with unstructured information, such as the description of errors.

With hybrid databases, structured and semi-structured information can be represented in one database.

4.2 Problems

Despite the enumerated advantages, two problems arise from the use of hybrid databases.

- Database systems have to be extended to exploit XML attributes within a database attribute. A prototype implementation for DB2 exists [Por99].
- An optimal structure for the hybrid database must be determined during the design process.

We subsequently concentrate on the second problem in the next section. We demonstrate how to find an optimal database design and suggest an algorithm for this task.

4.3 Design of hybrid databases

The following steps are used to design an optimal hybrid databases.

Step 1 We build a graph representing the hierarchy of the elements and attributes of the DTD.

Step 2 For every element/attribute of the graph, we determine a measure of significance w.

Step 3 We derive the resulting database design from the graph.

4.3.1 Step 1 — Determining a graph for a DTD

For a given DTD, we can create a corresponding graph representing the structure of the DTD (similar to the DTD graph suggested in [STH$^+$99]. We want to show the graph representation with an example. Therefore, we sort the elements and attributes into the graph accordingly to their hierarchical order.

Our sample DTD causes the graph in Figure 2. Thereby, recursions in the graph are represented by marked nodes. Furthermore, common sub-expression of the DTD are duplicated.

4.3.2 Step 2 — Calculation of weights

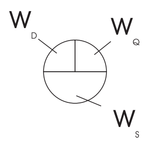
The next step is determining weights that specify the degree of relevance for every node of the graph (or the corresponding elements or attributes of the DTD). Thereby three kinds of information are included.

The first one is the *DTD structure*, but it can't deliver all information we need to suggest an optimal database schema. In our motivation, we showed that same or similar DTDs can be used in distinguishing applications in a different way. That's why, we also include the *available data* in our approach.

Furthermore, the available information in XML documents often doesn't coincide with the information the users search for. Therefore, we additionally include information of the *queries* into the estimation of the weights. In figure 3, we show a visualization of the three characteristics for one node.

All characteristics use a value between 0 and 1 for the degree of relevance. The following estimation specifies the calculation of the degree of relevance for every node based on these characteristics.

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



w_D	— weight, derived from the *existing XML data*
w_Q	— weight, derived from the *queries*
w_S	— weight, derived from the *DTD structure*

Figure 3: Node weightings

For the aim of clarity, we map the weights onto a grey-scale between $white = 0$ and $black = 1$ in our visualizations.

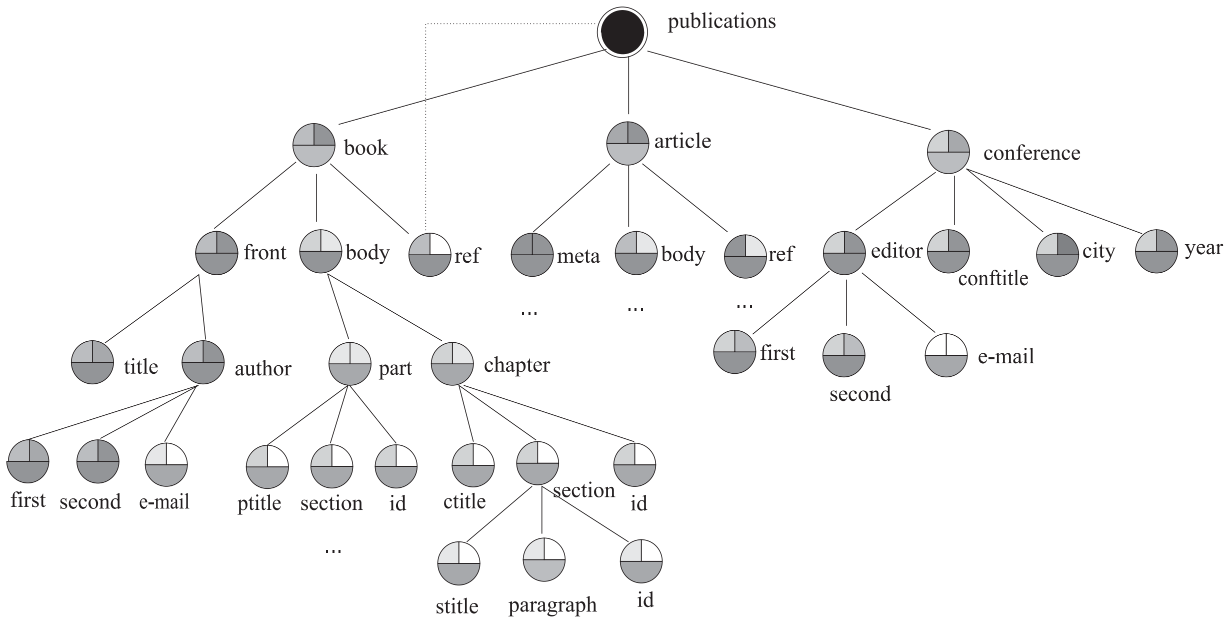


Figure 4: Visualization of the weights

Using three input parameters, we subsequently show how to estimate w_S.

S_H — position in the hierarchy:

This parameter can be calculated in the following way:

$$S_H = 1 - \frac{N_P}{N_D}$$

N_P — number of predecessors
N_D — maximal depth of the graph

S_A — exploitation of alternatives

DTD characteristic	S_A
mixed content	0
alternative on the same level exists	0.5
otherwise	1

S_Q — exploitation of quantifiers

DTD characteristic		S_Q
ELEMENT	no quantifier	1
	?	0.5
	+	0.75
	*	0.25
ATTRIBUTE	REQUIRED	1
	IMPLIED	0.5

Using these three structural characteristics, we can estimate the parameter w_S for every element and attribute of the DTD in the following way:

$$w_S = \frac{S_Q + S_A + S_H}{3}$$

Some information cannot be derived from a DTD. For instance, we do not know how often an element with the quantifier `'?'` or `'*'` is available in the documents. This is also true in cases where the attributes are `IMPLIED`. Therefore, the weight w_D is calculated:

$$w_D = \frac{D_A}{D_G}$$

D_A — number of documents containing the element/attribute
D_G — absolute number of XML documents

Furthermore, from DTDs we cannot derive which information is often requested and which information is of lower interest. Information about queries deliver this information and can help to find an optimal database design. The weight w_Q representing this information is calculated in the following way:

$$w_Q = \frac{Q_A}{Q_G}$$

Q_A — number of queries containing the element/attribute
Q_G — absolute number of queries

Summarizing these estimations, we get the following result:

$$w = \frac{1}{6}(S_Q + S_A + S_H) + \frac{1}{4}\frac{D_A}{D_G} + \frac{1}{4}\frac{Q_A}{Q_G}$$

Typical weights of the sample DTD for a bibliography application are represented in Figure 4. For fulltext the weights represented in Figure 5 can occur. It can be easily seen that the weights differ. Accordingly, the suggested database design also differs.

4.3.3 Step 3 — Deriving hybrid databases from the graph

We determine a limit specifying which attributes/elements are represented as attributes of the databases and which elements/attributes are represented as XML attributes. The limit can influence the level of detail of the resulting databases.

Next, we determine all nodes of the graph which fulfill the conditions that

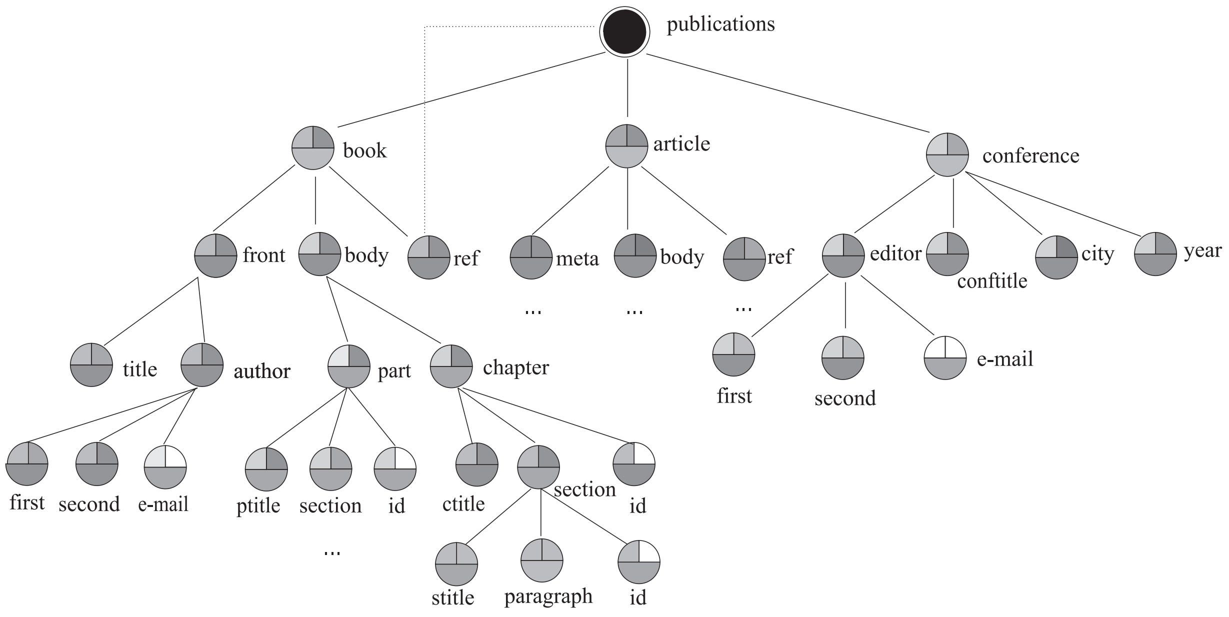


Figure 5: DTD graph with weighting 1

- the node is not a leaf of the graph
- the node and all its successors are under the level
- there exists no predecessor of the node that fulfills the same conditions

Finding an optimal limit is no easy task. It depends among other things on the response time of the query language for databases and XML.

All subgraphs consisting of these nodes and their successors are replaced by an XML attribute. If we apply this method onto the graph in Figure 4, we get as a result the graph in figure 6. Thereby, we used the DTD graph only for one reason: distinguishing the regular and irregular portions of the DTDs. Figure 6 shows the determined XML attributes. All the other elements and attributes of the graph are represented in the database according to the descriptions in the previous sections.

5 Related work

In [FK99], the authors suggest storing XML types in relational databases. The elements are stored all together within one relation. In that way, storing XML documents in a relational database is easy to realize. One advantage of the method is that the approach works for every document, also for documents without a DTD. The disadvantage of the approach is that queries are very difficult to realize. The query functionality of relational databases is lost. All queries that include more than one value can only be realized by joins.

Another approach for storing XML in relational databases is [STH$^+$99]. In contrast to [FK99], this approach results in large database schemas but provides better query performance. Additionally, optimization of elements results in separate relations.

In [Boe97], an approach is suggested for storing SGML documents inside of an object-oriented database. For performance reasons, it is impossible to transform all elements of the

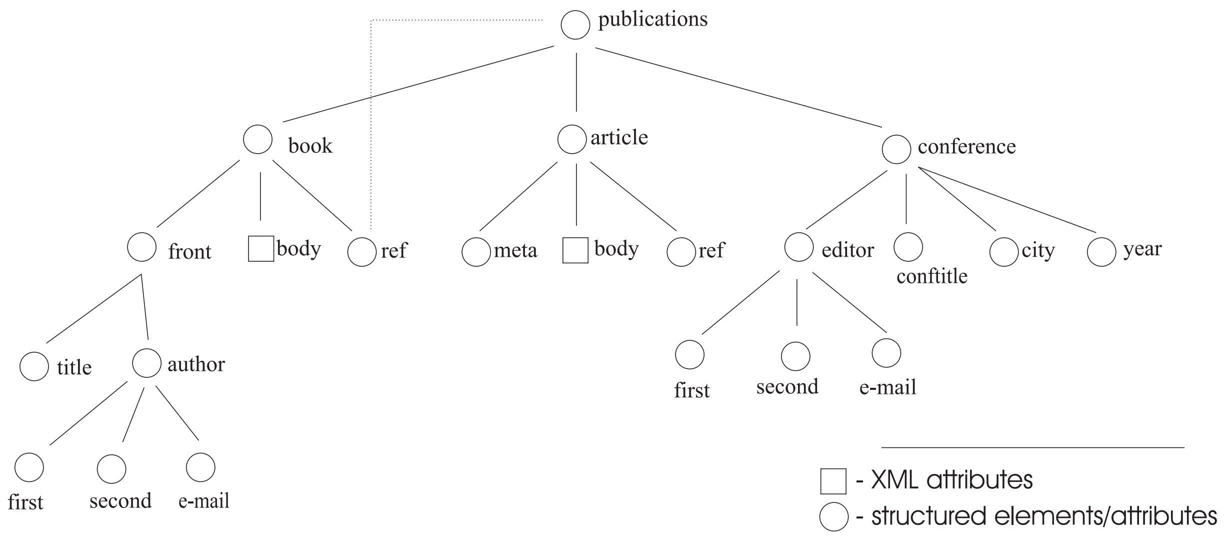


Figure 6: DTD graph with weighting 2

SGML structure into a one-to-one-relation of the object-oriented database. Only the “large elements” are mapped, the others are combined as “flat objects” that contain SGML. Thereby, the number of objects in the database is minimal. There are some similarities to our algorithm, but the approach didn’t automate the decision regarding which element structures remain as SGML fragments. The decision regarding how to represent elements is left to a DTD designer and database designer.

One application of [Boe97] is described in [DFM99]. Thereby, a used-car market application is realized with a hybrid database. A relational database is employed for the structured parts of XML documents and an XML attribute is used for all “irregular” portions. The design of the database is determined by the programmers, queries are realized as a combination of SQL and XQL on the application level.

None of the approaches take statistics on the document collections and query profiles into account. Most approaches that are based on a DTD accept only subsets of document type definitions for their database design.

6 Conclusion and future work

This paper suggests a mapping of DTDs onto object-relational and relational databases. To overcome the typical resulting problems (large schemas, sparsely populated), we suggested an algorithm for determining an optimal hybrid database schema.

Database systems like DB2/XML Extender support two methods to store XML data. The first one is mapping the data onto relational databases whereby the user has to specify how it shell be done. The second one is storing the data as XML data without changes. The designer takes the responsibility which representation is used. Thereby, the method suggested in this paper can support him/her.

First practical experiences exist in the context of the GETESS project (German Text Exploitation and Search System) on the domain tourism where we use object-relational databases (DB2) with XML attributes to store and query semi-structured information [SBB$^+$99].

Future work will include experiments with large document collections and associated DTDs. Thereby, the enumerated heuristics and their weightings have to be checked and validated.

Aspects, we don't focus on in depth in this article, are how to preserve document ordering, support update operations, allow restructuring of the document. Other important points for future investigation are query evaluations, optimization aspects, and automatic view definitions.

References

[Boe97] Klemens Boehm. *Verwendung objektorientierter Datenbanktechnologie zur Verwaltung strukturierter Dokumente*. PhD thesis, Technische Hochschule Darmstadt, 1997. (in German).

[Cla97] James Clark. Comparison of SGML and XML, December 1997. `http://www.w3.org/TR/NOTE-sgml-xml-971215`.

[Con98] World-Wide Web Consortium. Extensible Markup Language (XML) 1.0, 1998. `http://www.w3.org/TR/1998/REC-xml-19980210`.

[DFM99] Lars Dittmann, Peter Fankhauser, and Andre Maric. AMetaCar – a mediated eCommerce-Solution for the Used-Car-Market. In Ralf-Detlef Kutsche, Ulf Leser, and Johann Christoph Freytag, editors, *4. Workshop "Federated Databases", November 1999, Berlin, Germany, Proceedings*, pages 18–33. Technical University of Berlin, 1999.

[FK99] Daniela Florescu and Donald Kossmann. Storing and Querying XML Data Using an RDBMS. *Bulletin of the Technical Committee on Data Engineering*, 22(3):27–34, September 1999.

[Hul90] Guy Hulin. On restructuring nested relations in partitioned normal form. In Dennis McLeod, Ron Sacks-Davis, and Hans-Jörg Schek, editors, *16th International Conference on Very Large Data Bases, August 13-16, 1990, Brisbane, Queensland, Australia, Proceedings*, pages 626–637. Morgan Kaufmann, 1990.

[Por99] Beate Porst. Untersuchungen zu Datentyperweiterungen für XML-Dokumente und ihre Anfragemethoden am Beispiel von DB2 und Informix. Master's thesis, Universität Rostock, 1999. (in German).

[RKS88] Mark A. Roth, Henry F. Korth, and Abraham Silberschatz. Extended algebra and calculus for nested relational databases. *TODS*, 13(4):389–417, 1988.

[SBB$^+$99] Steffen Staab, Christian Braun, Ilvio Bruder, Antje Düsterhöft, Andreas Heuer, Meike Klettke, Günter Neumann, Bernd Prager, Jan Pretzel, Hans-Peter Schnurr, Rudi Studer, Hans Uszkoreit, and Burkhard Wrenger. GETESS — Searching the Web Exploiting German Texts. In M. Klusch, O. Shehory, and G. Weiss, editors, *Cooperative Information Agents III, Proceedings 3rd International Workshop CIA-99*, volume 1652. Springer Verlag, July 1999.

[STH$^+$99] Jayaval Shanmugasundaram, Kristin Tufte, Gang He, Chun Zhang, David DeWitt, and Jeffrey Naughton. Relational Databases for Querying XML Documents: Limitations and Opportunities. In *Proceedings of the 25th VLDB Conference, Edinburgh, Scotland*, pages 302–314, 1999.