Improving Storage Concepts for Semantic Models and Ontologies

Edgar R. Weippl[1]

Vienna University of Technology, A-1040 Vienna, Austria,
weippl@ifs.tuwien.ac.at

Abstract. Ontologies are more commonly used today but still little consideration is given of how to efficiently store them. The proposed approach is built on reliable and efficient relational database management systems (RDBMS). It can be easily implemented for other systems and due to its vendor independence existing data can be migrated from one RDBMS to another relatively easy.

1 Introduction

During the last couple of years ontologies moved into the center of interest in mainstream computer science research. With the Internet becoming a truly global information resource, the effort required to find the right information increased, even though the quality of search engines improved considerably.

The next big step is anticipated to be the integration of semantic information of electronically available resources which will allow searches to obtain much better results. The process of building the required ontologies can either be top-down or bottom-up.

The RDF-based approach, favored by Tim Berners-Lee, strives to semantically enrich each Web page and build ontologies by integrating all the semantic information. Topic Maps, in contrast, are usually regarded as top-down approach where occurrences are linked to topics once the Topic Map exists.

A prerequisite to building large ontologies is an efficient way of storing the required data. Today, it is generally agreed that ontologies evolve over time and require maintenance. Thus both retrieval and updates need to be handled efficiently by the storage system.

In this chapter we present an improved database schema to store ontologies. More specifically, our contribution is to

- propose an intuitive and efficient way of storing arbitrary relationships (Section 2.1).
- show that our database schema is well suited to store both RDF and Topic Maps (Section 2.2).
- explain why it is more efficient by comparing it to other approaches (Section 3).

2 Link-Based Schema

In this section we first explain the general idea of the improved database schema and provide an example of how concepts and relationships between them are stored. We then show how both RDF and Topic Maps can be stored, too.

2.1 Database Schema Based on a Link Table

The idea, first described in (Weippl et al., 2005), is based on an architecture that uses relational database. Tables are not linked to others *directly* with foreign keys or by using $n : m$ intermediary tables but via a single, generic association table referred to as the *link table*.

In the classical schema, adding an $n : m$ relationship between two tables requires creating a new intermediate table to resolve the $n : m$ relationship into a $1 : n$ and a $1 : m$ relationship . Our approach is to merge these intermediate tables into one link table which stores all relationships centrally.

The advantages of our approach are:

1. In contrast to classic E-R approaches, any relationship can be added without schema modifications. This allows to easily perform operations within transactions.
2. Tables and indices can be clustered to improve the speed of `join` operations with the central `link` table. In the classical model many $n : m$ relationships exist, therefore, cluster optimizations are far more difficult and less efficient.
3. Our approach allows retrieving relationships from the link table without accessing the data dictionary. Since the data dictionary is vendor specific, the classical approach requires modifying the application for each database system.
4. If n entities exist and $n : m$ relationships are to be established between all entities, the number of additional tables is $O(n^2)$, whereas our approach is $O(1)$ Of course this applies only to new relationships, not new tables.

Detailed explanations on the advantages can be found in (Weippl et al., 2005).

Figure 1 shows a simple database schema. Table ref1 contains the SQL statements of the following example. A new file type `Document (.doc)` is created with `OpenOffice`. An optional description is added and a relationship between the two topics is established (steps 1–4). In the same way occurrences can be linked to topics.

Steps 5–10 in Table ref1 show how reification (Figure 2) can be implemented using our schema.

Our concept differs from to other approaches (Section 3) by using separate tables to store different types of entities but one central link table for all relationships. The data-centric approach, which we also refer to as the 'classical' way, uses one table for each $n : m$ relationship. The structure-centric approach stores everything in one table (such as an RDF triple store).

The advantage compared to the data-centric approach is that we require fewer changes of the database schema during normal database operations. Adding a

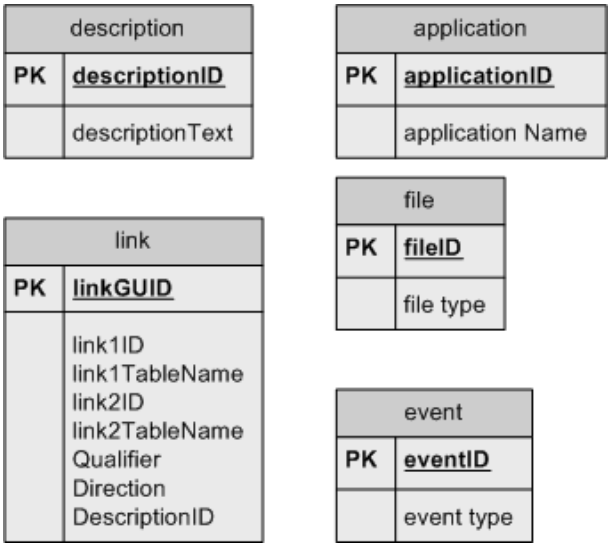

Fig. 1. The database schema to store the information as given in Table ref1.

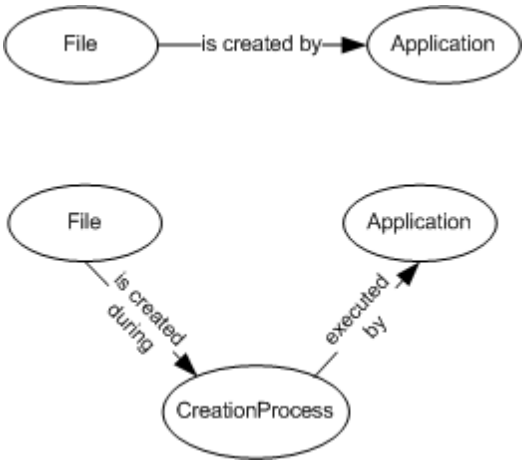

Fig. 2. Reification

new type of relationship — by far the most common operation — requires no schema modification. The structure-centric approach has the same advantage but suffers from another drawback. Since everything is stored in a single (or very few) tables this table will quickly become very large and thus access slower. Numerous self-joins, which will be required, also have a negative impact on performance.

Step	SQL Command
1	`INSERT INTO file VALUES (1, 'Document (.doc)')`
2	`INSERT INTO application VALUES (10, 'OpenOffice')`
3	`INSERT INTO description VALUES (90, 'save as operation')`
4	`INSERT INTO link VALUES (111, 1, 'file', 10, 'application', 'assocrl', '1', 90)`
5	`INSERT INTO event VALUES (42, 'save as')`
6	`INSERT INTO link VALUES (112, 1, 'file', 42, 'event', 'assocrl', '1', 91)`
7	`INSERT INTO link VALUES (113, 42, 'event', 10, 'application', 'assocrl', '1', 91)`
8	`DELETE FROM link WHERE linkGUID=111`
9	`DELETE FROM description WHERE descriptionID=90`
10	`INSERT INTO description VALUES (91, 'reification')`

Table 1. A relationship between a document and an application is stored (steps 1–4). An example for a reification is given in the following steps.

2.2 Storing Topic Maps and RDF

Even though the structure-based approach is slower during retrieval, it may make sense to implement it in a very dynamic environment where new entities, new relationships and even new types of relationships are often created. These characteristics typically apply to semantic environments such as RDF or Topic Maps. Modifying the aforementioned link-based architecture we show that the relational storage model as proposed by (Widhalm and Mück, 2002) can be optimized in several ways helping to improve the performance and reduce the complexity of the database schema.

First and foremost we can reduce the number of tables used without the loss of data or metadata (Figures 3 and 4). By using `qualifiers` in the link table we can combine tables such as `basename`, `sortname`, `dispname` and `topname` into one table called `name`. The qualifier attribute in the link table contains information whether the name is used as basename, sortname, etc.

Following the XTM standard[1] we also no longer need the table `facet`. The link that connects topics and associations stores the `association` role as `qualifier`, not in a separate table. In the same way we can avoid separate tables for `fvalue, locationstype, nonconforming` and `cassign`.

Since 'everything' is a topic we do not need to explicitly store this information in a table. Instead, we propose to create a view that contains all the information (`create view ... as select from ... UNION selection from ...`).

The main difference between RDF and Topic Maps that is relevant to storing information is that RDF only supports relationships between two entities — RDF uses nodes and arches to build graphs of concepts and relationships between them. This makes storage much easier and the simplest approach is to store

[1] http://www.topicmaps.org/xtm/1.0/

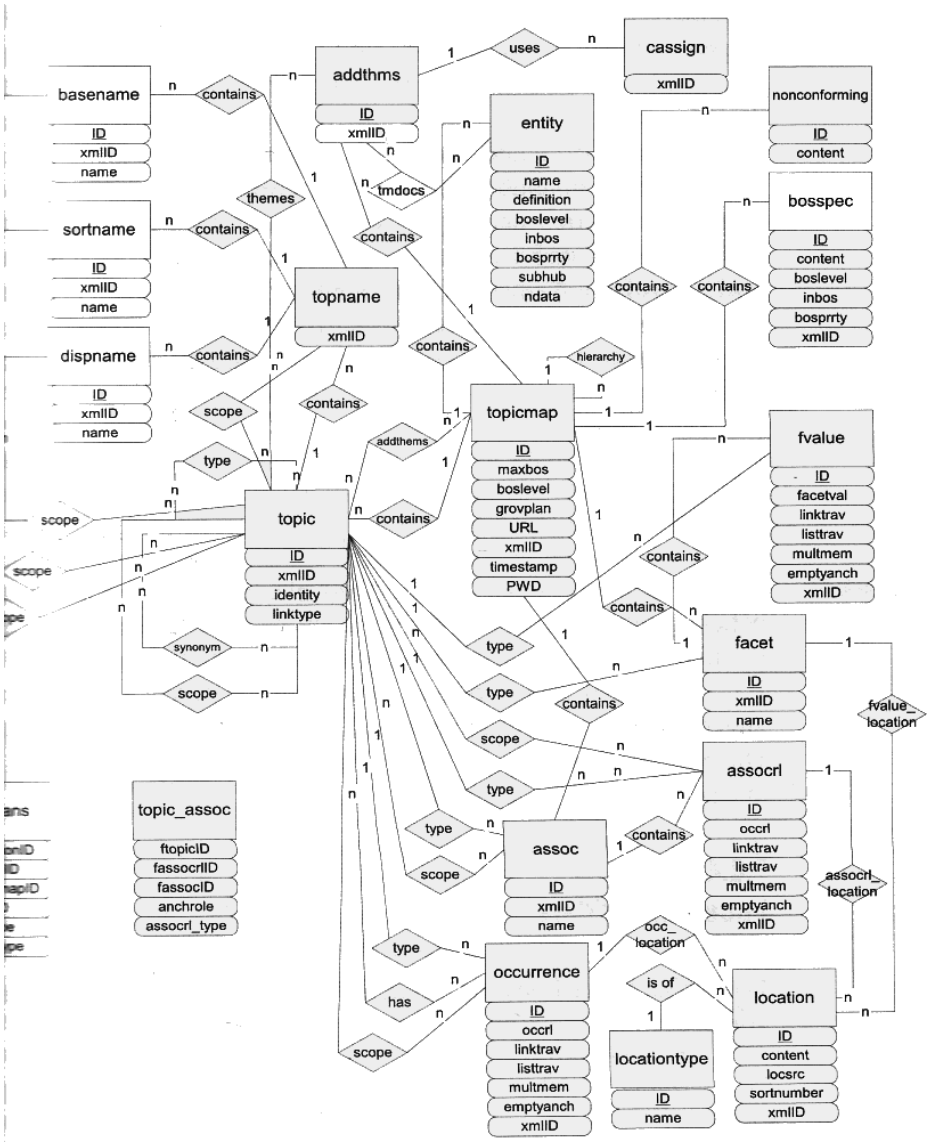

Fig. 3. Storing Topic Maps in an RDBMS (Widhalm and Mück, 2002).

RDF triples in the form (s, p, o) (subject, predicate and object) (R. Agrawal and Xu, 2001).

However, RDF can be stored similarly as Topic Maps either using the 'pure' link-based approach (Section 2.1) or by modifying it analogous to what we showed for Topic Maps. All four major differences between RDF and Topic Maps can be handled by the link-based approach:

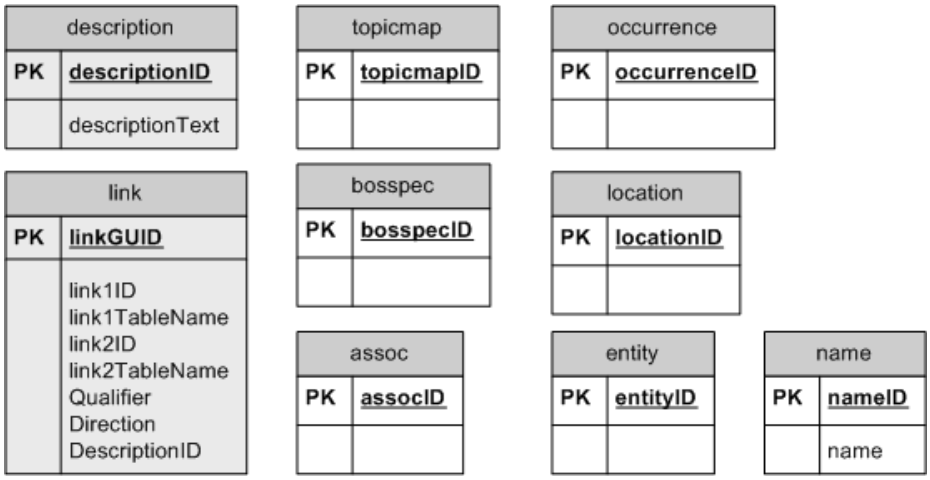

Fig. 4. By storing all relationships in the `link` table together with a qualifier, fewer tables (compare to Figure 3) are needed but all advantages as described in (Widhalm and Mück, 2002) are retained.

1. In RDF relationships can only be established between two resources whereas Topic Maps supports relationships between any number of topics. The link table supports an arbitrary number of links.
2. In RDF relationships are directed and only valid for one direction. In most cases this requires creating a redundant second and inverse relationship. In the link table an attribute is used to store the direction.
3. In contrast to Topic Maps, RDF does not support scopes which makes it difficult to create large ontologies by combining existing smaller ones. If scopes are required, a table (scope) needs to be added. By linking the appropriate scope via the link table, scopes can be easily handled.
4. In RDF reification is necessary if additional information needs to be attached to a relationship later on. This is not necessary for Topic Maps since everything is already reified. Reification can be performed efficiently as shown before.

3 Evaluation of Other Storage Concepts

In this section we briefly look at three systems that store personal information and strive to provide semantically enriched retrieval capabilities.

We then look at existing solutions to organizing a semantic data store.

- XML databases (Section 3.2)
- data-centric approach with relational databases (Section 3.3)
- structure-centric approach with relational databases (Section 3.4).

3.1 Storing Personal Digital Information

Vanevar Bush's vision of the Memex (Bush, 1945) — a paper that almost everyone cites when writing about semantically enriched information storage — is the basis on which projects such as Microsoft's *MyLifeBits* (Gemmel et al., 2002) or the *SemanticLIFE* project (Ahmed et al., 2004) build.

The authors want to build a personal digital storage that records all documents, emails, photos, videos, etc. of an individual.

MyLifeBits focuses on storing digital content in a database; unlike SemanticLIFE it does not primarily aim to semantically enrich the stored data. Instead MyLifeBits relies on future improvement of search engines and desktop search solutions. The focus of SemanticLIFE is building ontologies and discovering relationships between existing data items.

Haystack (Adar et al., 1999) is a platform to visualize and maintain ontologies. The system is designed to flexibly define interactions and relationships between objects. The focus lies on the quality of the retrieval process and not on storing data.

While both systems inherently address issues of storing ontologies they do not focus on an efficient storage concept. MyLifeBits assume that MSSQL Server will provide all the needed functions without providing details on the database schema used.

3.2 XML in Oracle

Both RDF and Topic Maps (XTM) can be encoded in XML. Major relational database management systems (RDBMS) now offer ways of storing XML content.

With Oracle9i a new data type `XMLType` was introduced. The idea is that XML data can be stored in the database that natively supports XML. Previously XML data was often stored in an unstructured way using BLOBs or CLOBs; CLOBs are Character Large Objects and compared to BLOBs have an advantage when text is required to support different code pages.

When an XMLType is used Oracle provides several features that are specific to XML data. for instance, XPath[2] can be used to query the stored data directly. Moreover, XSL transformations and schema validation. However, whenever XML data is modified, the XMLType requires an XML syntax check; obviously this tasks requires processing time. Dillon (Dillon, 2005) reports that inserting data in an XMLtype results in an approximately 22-fold increase of processing time, even if the XML document is a very simple document:

```
<?xmlversion="1.0"?>
<testTag/>
```

[2] To retrieve, for instance, the last name of a person following XPath query could be used: `select extractValue(OBJECT_VALUE, '/Person/FirstName') from xmldocs;`

Oracle offers another option to store XML documents. Using the object-relational approach XML documents are decomposed into data elements when inserted into the database. The data is stored in a relational model and can be queried accordingly[3]. In addition, the original XML document can be re-assembled automatically. The obvious drawback is that the XML schema has to be registered with the database.[4]

Storing XML in an RDMS such as Oracle is a good idea if many XML documents need to be stored since the database is optimized to handle a large number of individual documents. A Topic Map, however, is a single and very large XML file; thus the database cannot handle updates to this document efficiently.

3.3 Data-Centric Approach

One approach, also known as data centric approach, is often mentioned in context with mapping XML documents to relational databases (Kuckelberg and Krieger, 2003; Bourret, Bourret, 2001; Mittermeier, 2003). With respect to ontologies, the process can be described as follows.

The first step is to identify the types of concepts and their properties that are to be stored in the ontology. Then, these types of concepts are mapped to according tables in a traditional RDBMS, with the previously identified properties being the fields of the tables. Finally, the instances of the classes can be inserted into the tables as rows, with one row representing one instance of a concept. This procedure is the same for subjects, relationships and all other data model entities defined by the respective standard.

In addition, several 'auxiliary' tables are needed to keep track of whether a certain table maps to a subject or to a relationship etc. This leads to the situation that the database is actually split into two 'virtual layers': the virtual 'schema layer' consists of the auxiliary tables that keep track of all classes in the ontology, whereas the virtual 'data layer' contains the tables created as instance containers for specific classes.

Such a data centric approach was for instance originally followed by the Sesame ontology framework (Broekstra et al., 2001, 2002) in conjunction with a PostgreSQL database. Figure 5 shows the setup of the Sesame data centric object-relational mapping.

There are two advantages that can be exploited with the data centric approach. First, query answering as well as inserting, removing and updating instance of classes is extremely inexpensive and straightforward, as there is virtually no difference to traditionally designed databases. All manipulations concerning instances are in effect nothing more than executions of data manipulation commands as they are natively provided by all RDBMS.

Second, some RDBMS like PostgreSQL offer built-in object-relational features that can be used directly for modelling class-subclass relationships etc.

[3] For instance, `select x."XMLDATA"."FirstName" from xmldocs x;`

[4] Oracle10g provides the stored procedure `DBMS_XMLSCHEMA.REGISTER_SCHEMA`, making frequent modifications of the XML schema difficult.

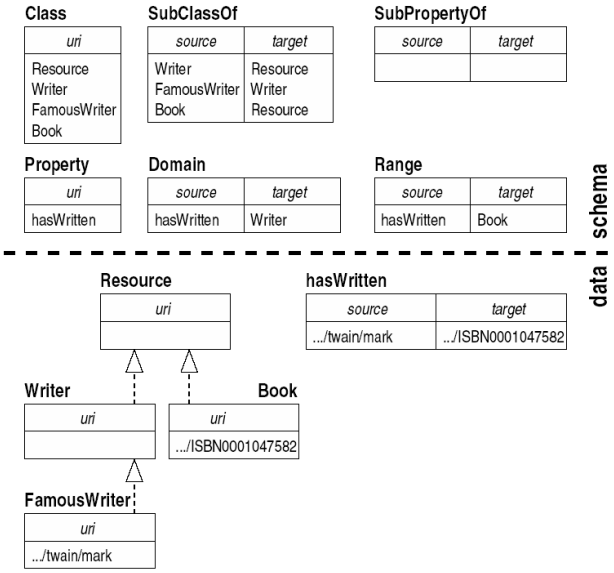

Fig. 5. Data-centric approach of Sesame (Broekstra et al., 2001).

PostgreSQL databases offer for instance the possibility to create subtables that are connected to their parent tables through transitive relationships. This allows for creating a table for a certain class and according subtables (for subclasses of that class). The same is true for properties and subproperties, accordingly.

The main drawback of the data centric approach is that changes to the class hierarchy in an ontology are extremely expensive, as they require creating new entities in the database. For every new class (and also subclass) that is to be inserted into the ontology, a respective table has to be created, even if only a small number of instances is present. This means that changes to the class hierarchy always require data definition commands to be performed, which are expensive in almost any RDBMS.

3.4 Structure-Centric Approach

The second approach is also known as structure centric and is equally popular among Topic Map and RDF implementations. As it is the case with the data centric approach, persistency is eventually provided by a traditional RDBMS, but usually without requiring object-relational features. Opposed to the first approach, the key idea here is to map the finite number of data model concepts to according structures (tables) in the relational database. Again, the process has also been described for XML documents (Kuckelberg and Krieger, 2003; Bourret, Bourret, 2001; Mittermeier, 2003) but has also been specifically implemented for both Topic Map and RDF applications.

As shown in detail in Section 2.2, the Topic Map data model offers a small number of built-in concepts, like Topic, Association, Occurrence, Scope etc., whose properties are well defined. In contrast to the actual classes and instances they represent, the number and design of these built-in concepts are static (as they are standardized). Therefore, it is a straightforward task to create corresponding structures in a RDBMS and map the concepts to these structures in such a way that in the end there is one table for all topics, one table for all associations, etc. Various examples of this implementation for Topic Maps exist, e.g. (Kiyakov et al., 2001; Widhalm and Mück, 2002).

With respect to RDF, the data model consists basically only of statements, with each statement including a subject, an object and a predicate. This means that for a naive approach only one single table (with three corresponding text fields containing the respective URIs or literals) is needed to express a complete RDF graph. Due to the layout of their tables, databases configured this way are therefore commonly referred to as triple stores. They are certainly a very elegant solution for ontology persistence and are probably one of the main reasons that RDF/OWL has gained significant popularity among ontology developers. Also, many variations and improvements over the naive approach exist, mainly in order to achieve high levels of scalability.

The first advantage of the structure centric approach is its ability to allow for inexpensive, frequent changes of instance data as well as of schema information (class hierarchies). Since all assertions, including hierarchical relations, are broken down to the level of single statements, no artificial distinction between "schema layer" and "data layer" has to be made. This allows not only for representing frequently changing ontology hierarchies, but also for efficient incremental incorporation of large datasets, as no structural changes of the underlying database schema are required.

The second advantage of structure centric ontology representation is commonly found to be reported for dedicated triple stores, but also applies for Topic Map representations. Due to the fixed, rather simple architecture of the database, scalability optimizations are easy to apply, enabling the efficient storage of millions of concepts and relationships.

One main disadvantage of the structure centric approach (in the case of RDF triple stores) is encountered when retrieving statements for answering ontology queries. In order to evaluate a condition that not directly addresses the URIs or literals of the statements to be retrieved, the table containing the statement triples has to perform one or more self-joins, an operation which is expensive for large datasets (Kuckelberg and Krieger, 2003; Alexaki et al., 2001). Such large datasets must be considered to occur frequently, as all information of an ontology is stored within a single triple table. It is therefore not uncommon for such a table to contain millions of triples, which have to be compared to each other even several times, depending of the nature of the query to be answered. Although various optimization efforts try to limit the negative effects of storing triples in a single table, generally worse query answering performance has to be expected compared to the object-relational approach.

4 Conclusion

We proposed an improved way of storing ontologies in a relational database so that changes of hierarchies and relationships between tables can easily be added without schema modification. The advantages of our approach are:

1. Most modifications require no data-definition language (DDL) statements the cannot be executed within a transaction.
2. Tables and indices can be clustered to improve the speed of joins with the central `link` table.
3. Our approach is vendor-independent as no metadata on relationships need to be retrieved from the data dictionary.

In addition, we showed that both Topic Maps and RDF can be stored efficiently using our database schema.

5 Acknowledgment

This work was performed at the Research Center Secure Business Austria funded by the Federal Ministry of Economics and Labor of the Republic of Austria (BMWA) and the federal province of Vienna.

Bibliography

Adar, E., D. Karger, and L. A. Stein (1999). Haystack: Per-user information environments. In *Proceedings of the Conference on Information and Knowledge Management.*

Ahmed, M., H. H. Hanh, S. Karim, S. Khusro, M. Lanzenberger, K. Latif, M. Elka, K. Mustofa, N. H. Tinh, A. Rauber, A. Schatten, N. M. Tho, and A. M. Tjoa (2004, September). Semanticlife — a framework for managing information of a human lifetime. In *Proceedings of the 6th International Conference on Information Integration and Web-based Applications and Services (IIWAS).*

Alexaki, S., V. Christophides, G. Karvounarakis, and D. Plexousakis (2001). On storing voluminous RDF descriptions: The case of web portal catalogs. In *Proceedings of the 4th International Workshop on the the Web and Databases.* ICSFORTH.

Bourret, R. Xml-dbms. http://www.rpbourret.com/xmldbms/readme.htm.

Bourret, R. (2001). Mapping dtds to databases. Technical report, XML.com. http://www.xml.com/lpt/a/2001/05/09/dtdtodbs.html.

Broekstra, J., A. Kampman, and F. van Harmelen (2001). *Semantics for the WWW*, Chapter Sesame: An Architecture for Storing and Querying RDF Data and Schema Information. MIT Press. http://www.cs.vu.nl/ frankh/postscript/MIT01.pdf.

Broekstra, J., A. Kampman, and F. van Harmelen (2002). Sesame: A generic architecture for storing and querying rdf and rdf schema. In *ISWC 2002.* ISWC. http://www.openrdf.org/doc/papers/Sesame-ISWC2002.pdf.

Bush, V. (1945). As we may think. *The Atlantic Monthly 176*(7), 101–108.

Dillon, S. (2005, April). Which storage xml? Oracle Magzine. http://www.oracle.com/technology/oramag/oracle/05-mar/o25xmlex.html.

Gemmel, J., G. Bell, R. Lueder, S. Drucker, and C. Wong (2002, December). Mylifebits: Fulfilling the memex vision. In *ACM Multimedia '02*, pp. 235–238.

Kiyakov, A., K. Simov, and M. Dimitrov (2001). Ontomap: Ontologies for lexical semantics. Technical report, OntoText Lab, Sirma AI EOOD. http://www.ontotext.com/publications/ranlp01.pdf.

Kuckelberg, A. and R. Krieger (2003). Efficient structure oriented storage of xml documents using ordbms. Technical report, RWTH Aachen.

Mittermeier (2003). Naiv nativ. *iX 42*(8).

R. Agrawal, A. S. and Y. Xu (2001). Storage and querying of e-commerce data. In *Proceedings of VLDB 2001*, Rome, Italy. http://www.vldb.org/conf/2001/P149.pdf.

Weippl, E., M. Klemen, M. Linnert, S. Fenz, G. Goluch, and A. M. Tjoa (2005). Semantic storage: A report on performance and flexibility. In *submitted to DEXA 2005.*

Widhalm, R. and T. Mück (2002). *Topic Maps: Semantische Suche im Internet.* Springer Verlag. ISBN 3540417192.