The Semantic Desktop: A Semantic Personal Information Management System Based on RDF and Topic Maps

Edgar R. Weippl, Markus Klemen, Stefan Fenz, Andreas Ekelhart, and A Min Tjoa

Vienna University of Technology, A-1040 Vienna, Austria
weippl@securityresearch.at
http://www.securityresearch.at

Abstract. Desktop search tools are becoming more popular; they allow full text searches using inverted indexes. Yet, the amount of locally stored data that they have to deal with is increasing rapidly. A different approach is to analyze the semantic relationships among collected data and thus preprocess the data semantically. The goal is to allow searches based on relationships among various objects rather than focusing on objects' names. This would allow for searches far more sophisticated than those based on full text analysis. We introduce a database architecture based on an existing software prototype that is capable of meeting the various demands of a semantic information manager. This architecture is also capable of storing and querying RDF and RDF schemata. Moreover, RDF is used as a key part of the technology. Therefore, in this scenario, RDF is used not only to enrich the Web with machine-processable semantics, but also to incorporate it into a kind of Semantic Desktop Search Engine. In this paper, we describe the underlying technology of this research project.

1 Introduction

More and more data is accumulating on personal computers these days. People store their journals, time managers, contact data, photos, and other documents on their computers. Despite all efforts, thus far no search tool has been created that allows searches based on semantic connections. What is interesting is that most current approaches focus on enriching the World Wide Web semantically. Our approach focuses in on the domain of a single user who stores and retrieves data on one or more computer systems using semantic enrichment.

Although it is accordingly situated somewhere between RDF-based or Topic Map-based Semantic Web projects, such as, Sesame [8] and and personal lifetime data storage projects, such as, MyLifeBits [13] or SemanticLife [3]the approach and underlying architecture differ fundamentally from either of these concepts.

For retrieval, our approach focuses on the relationships among various local data-objects (such as, photos, e-mails, graphics, and text files) and events (opening a text file, receiving a phone call, sending an e-mail) rather than relying on

M. Collard (Ed.): ODBIS 2005/2006, LNCS 4623, pp. 135–151, 2007.
© Springer-Verlag Berlin Heidelberg 2007

the names of these objects. Our intent is to allow for more human-like retrieval processes by adding semantic metadata to the data collections. For example, instead of finding a text file based on its name, a semantic search would allow a context-aware query, for instance, *I don't know the filename but I know I created it when I was talking to Jim on the phone about a week ago.*

Our prototype collects raw data from multiple sources such as the operating system's file events or user events from Microsoft Outlook via agents. At the file-system level, we receive data identifying the applications that have accessed (read or written) a particular file. The user can select which applications or directories to monitor, typically these are local office applications and user document folders. For instance, we can store the information that Word.exe has saved a specific file document.doc on the disk at a certain time and date. From Microsoft Outlook we can gather information on emails, contacts, and calendar items. We also store information on the specific computer used (to differentiate between laptops and workstations) and the user ID. Planned are additional data collectors that can integrate incoming and outgoing telephone calls (via CTI or serial printer ports) as well as facsimiles, GPS data, and EXIF[1] metadata from digital camera images.

Based on the vast amounts of data accumulated, a semantic enrichment engine (SEE) is implemented that uses the data and derives information from it to build semantic databases for human users. Clearly, the usefulness of the system as a whole depends on the quality, speed, and versatility of the semantic databases and on the capabilities of the semantic enrichment engine. In this paper, we will focus on the underlying database schema and propose a database architecture that provides the foundation for semantic analysis. There are certain requirements for such a database:

- *Flexibility:* A database for semantic storage must be highly flexible. It must be able to store heterogeneous data from various sources including e-mail systems, file systems, date books, telephones, and GPS modules. Defining new relationships between existing entities will be a common task.
- *Compliance:* The database should be compliant to emerging Semantic Web standards such as RDF or Topic Maps.
- *Backwards compatibility:* All enhancements to the database must be backwards compatible. Modification of the database schema should occur only rarely.
- *Speed:* The database must perform well at high speeds due to the high volume of processed data.
- *Scalability:* The database design should allow for up-scaling of the database with no significant performance loss.

More specifically, our contribution:

- provides an overview of the architecture of our Semantic Desktop Project prototype
- proposes an intuitive and efficient method for storing arbitrary relationships (Section 3.2).

[1] `http://www.exif.org/`

– shows that our database schema is well suited to store both RDF Metadata and Topic Maps (Section 3.3).
– explains why it is more efficient in comparison with other approaches (Section 4).

2 The Problem

An increasing capacity for data storage enables people to save virtually their whole life digitally in various file formats or databases-photos, videos, e-mails, address databases, etc. Available personal programs to store and manage these files usually offer searches either via file system hierarchies or via keywords or full text search (in cases where the file contains text data). *Filesystem hierarchies* not well suited since it is often not possible to make precise attributions to a single, specific folder [11]. *Keywords* are commonly either based on the file name or must be typed in manually. Manual keyword input is cumbersome, time consuming, and subject to the *Production Paradox* [10]—people will simply not do it since they see no immediate advantage. *Fulltext-engines*, on the other hand, are useful for text-based documents only. Integrating photos and music into full-text-based systems is difficult and an area of ongoing research.

Apart from that, people tend to forget names of specific objects. It is often easier to remember the *context* of a situation in which a specific file was created, modified, or viewed, especially with reference to a *timeline* ("I remember I just got an e-mail from Mike when I was working on that document").Semantic enrichment of automated data-gathering processes is a useful tool to complement this human, relational way of thinking, rather than thinking in keywords or tags.

3 The Semantic Desktop Project

At the first phase, Blackman should help the user to scan the computer, queried by Blackman query language (see [12]), for certain data files and an important part for this task is the integration of Microsoft Outlook 2003 and the collection of data hosted there. A big part of information, a user is producing, is located at the e-mail-client and so Blackman works in a first try with Microsoft Outlook 2003, to extract the following elements:

– e-mails
– calendar entries
– contact entries

The extracted data will be saved at database to make a future query- and rule-creation possible. Due to the modular structure of Blackman it is not difficult to integrate further watchers for additional e-mail-clients produced by other vendors than Microsoft. The second part of the Blackman project should monitor the file system and network activity to gather as much user data as possible. An example: If the user is opening a file, Blackman should recognize this, to create

an entry of this event at database. This action will be represented by an event which could be enriched with some information like time, location and certain other circumstances at which the file was opened.

In a next step a 'Semantic enrichment engine' should make the collected data useful to the user. The engine should implement certain ontologies which can be used, to enrich collected data semantically, for purposes like 'Personal organization', 'Security', 'Visualization' or any other usage where data of user's behavior is needed.

So how could this data be used to make the organization of user's life much easier?

At this phase of Blackman it is necessary that the user is planning his day, is administering his contacts and is storing his e-mails with Microsoft Outlook 2003. With this precondition Blackman is able to reconstruct what and when the user is doing something, to reconstruct users daily life.

The following listing describes a few sample use-cases to make the ideas above more understandable:

- When the user is participating at a meeting from 10:00 to 12:00 and is working at a certain document for a defined duration it is highly possible that this document has something to do with the meeting. If there is a meeting next week with the same participants and a similar topic, Blackman should collect automatically all relevant documents and make them available to the user before the meeting starts. This would be a use-case for a specific business usage.
- In many companies it is normal that documents, even confidential documents, are sent by e-mail to the desired recipient(s). This could be a security approach for Blackman; the semantic enrichment engine could be implemented in a way that it 'knows' which documents are confidential. There are several ways how Blackman could classify a document automatically as confidential. One possible approach would be that Blackman is looking, when user receives a document, in the address book entries for sender's position within the company. Blackman also looks on the list of recipients which can be found in the header of any e-mail. If, for instance, user's department chief is the sender and the mail was sent only to one person it is highly possible that this document is confidential. From this point Blackman 'knows' that this document is confidential and monitors every action which has something to do with the, as confidential classified, document. At another day the user wants to forward this document unintended to all co-workers, due to analysis of e-mail header and content Blackman will recognize that, and fires up an alarm.

  A different security approach would be the detection of abnormal user activity. Blackman records almost every action taken at users machine. If, for instance, due to a evil worm, abnormal outgoing network traffic is generated, Blackman could block and alert this traffic, to ensure users data integrity and security. Therefore Blackman could be implemented in a specific way, to provide similar functions as a 'Intrusion Prevention System'.

– If Blackman is installed organization wide it is possible to track document changes to enable the creation of a work flow visualization. Not only the work flow within the organization is tracked, due to e-mail monitoring also contacts to external actors are recognized by Blackman and could be merged with the internal work flow. The creator or owner of a document could see what is happening with 'his' document and who sends it to whom.

  The surveillance of network- and e-mail traffic enables Blackman also to build up a visualization of social networks.

Recapitulating, Blackman should help the user to organize his data, which could be realized by recording his daily life behavior. Based on automatically or manually created rules the collected data will be enhanced with semantic data to provide, through Blackman, a practical benefit to the user. The whole data gathering process is happening in background, to ensure that the user has not to 'fight' with an additional system on this machine.

The Semantic Desktop project goes far beyond typical full-text analysis search engines by automatically enriching collected data with semantic context that can be used for retrieving it more easily than without this context.

Our prototype was developed in DotNet and Java, and consists of five major development components:

1. *Request Handler:* The Request Handler consists of various modules to process external data sources. It is explained in more detail in Section 3.1.
2. *Semantic Storage:* Storing semi-structured, highly interconnected data requires data models that take these characteristics of semantic environments into consideration. In this paper, we thoroughly explain how our approach satisfies those requirements.
3. *Semantic Enrichment:* Semantic Enrichment is crucial for the usability of a semantic information management system.
4. *Querying Interface:* The Querying Interface is another critical element. We develop an interface that is compatible with OWL while still providing easy and secure access to the specific needs of a personal desktop information system.
5. *Client Application:* Currently, we have a prototype client in use written in DotNet. A Java-based Webclient is planned after the DotNet client is released and sufficiently stable.

In this paper, we will focus on the Semantic Storage development area. We introduce an improved database schema and provide examples for how concepts and relationships are stored among the databases. We then show (Sections 3.3 and 3.4) how both RDF and Topic Maps can be stored efficiently.

3.1 Request Handler

We distinguish four types of data input channels: (1) *Native Data Pipes* (2) *XML-based data exchange* (3) *SOAP request broker* (4) *HTTP request broker*

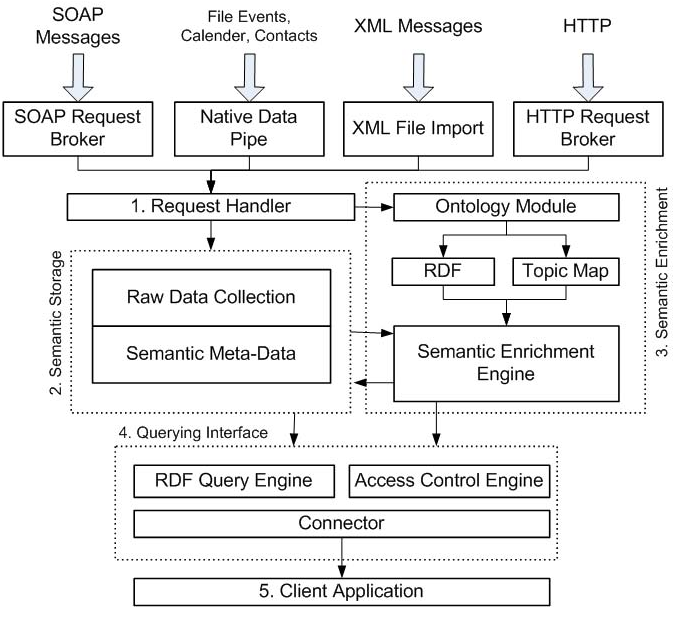

Fig. 1. System architecture: data is collected from various sources and stored in the raw data collection. Subsequently, the semantic enrichment engine (SEE) analyzes the data based on ontology guidelines and RDF or Topic Map based rules and adds links between recorded data items.

Native Data Pipes: Currently we have three data pipes. (1) Outlook Data Pipes for Microsoft Exchange Server, to access calendar entries, contacts, and e-mails; (2) an OS File Data Pipe, which is hooked directly into the I/O system of Windows 2000/XP/2003; and (3) a Network Traffic Data Pipe, which monitors network traffic for both URL requests and for tracking visited websites.

XML-Based Data Exchange: We use this module for research studies comparing Unix-based semantic data collection with the Windows-based variants. The idea behind this is to develop universally valid semantic statements, which may be used in both Windows-based and Unix-based environments. We are currently collecting data for semantic analysis on both Windows and Unix machines and we expect interesting results within the next half year.

SOAP Request Broker and HTTP Request Broker: Both modules are in the early stages but will facilitate the networking of various client machines to build a unified personal information management system. This will be an important part of the project since more and more users are working on more than one computer and therefore could profit from a system that would allow the interconnection of these devices.

3.2 Semantic Storage

Our prototype, first described in [17], is based on an architecture that uses relational databases. Tables are not linked to others *directly* with foreign keys or by using $n : m$ intermediary tables, but instead, 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 that stores all relationships centrally.

The advantages of our approach are:

1. In contrast to classical E-R approaches, any relationship can be added without schema modifications. This allows for easy performance of 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, multiple $n : m$ relationships exist, therefore, cluster optimizations are far more difficult and less efficient.
3. Our approach permits retrieval of 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 to new tables.

Detailed explanations on the advantages can be found in [17].

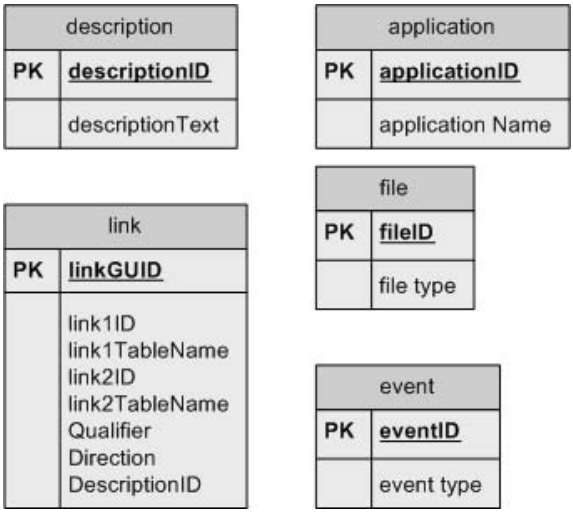

Fig. 2. The database schema to store the information as given in Table ref1

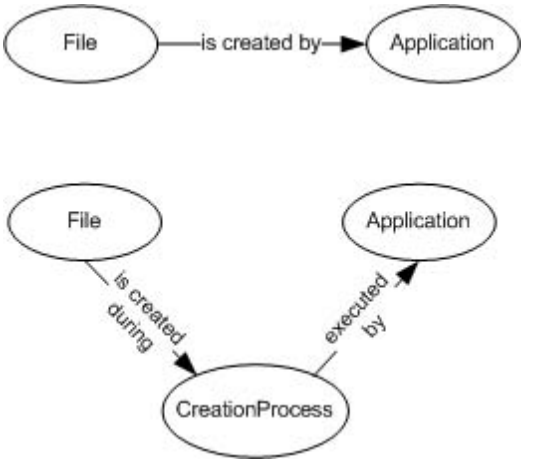

Fig. 3. Reification

Figure 2 shows a simple database schema. Table 1 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.

Reification is an important process for a semantic system. It is highly probable that a semantic analysis module will initiate reification while processing collected raw data. Steps 5-10 in Table 1 show how reification (Figure 3) can be easily implemented using our schema.

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.

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')`

Our concept differs from to other approaches (Section 4) 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" method, 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 of our approach as compared to the data-centric approach is that we require fewer changes of the database schema during normal database operations. Adding a new type of relationship—a very common operation in semantic systems-requires no schema modification. The structure-centric approach has the same advantage but suffers from a different drawback. Since everything is stored in a single (or very few) tables, this table will quickly become very large and thus be slower to access. Numerous self-joins, which will be required, also have a negative impact on performance. Moreover, only general purpose database indexes (B-trees) can be used. Our approach, in contrast, permits defining Bitmap and Function-based Indexes[2] that are extremely efficient in some cases and completely useless in all other cases.

3.3 Storing Topic Maps

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 created frequently. 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 [19] 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 4 and 5). 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 on whether the name is used as basename, sortname, etc.

Following the XTM standard[3] we also no longer need the table `facet`. The link that connects topics and associations stores the `association` role as `qualifier`, rather than 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 creating 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 RDF triples in the form (s, p, o) (subject, predicate, and object) [2].

[2] Using an Oracle database.

[3] `http://www.topicmaps.org/xtm/1.0/`

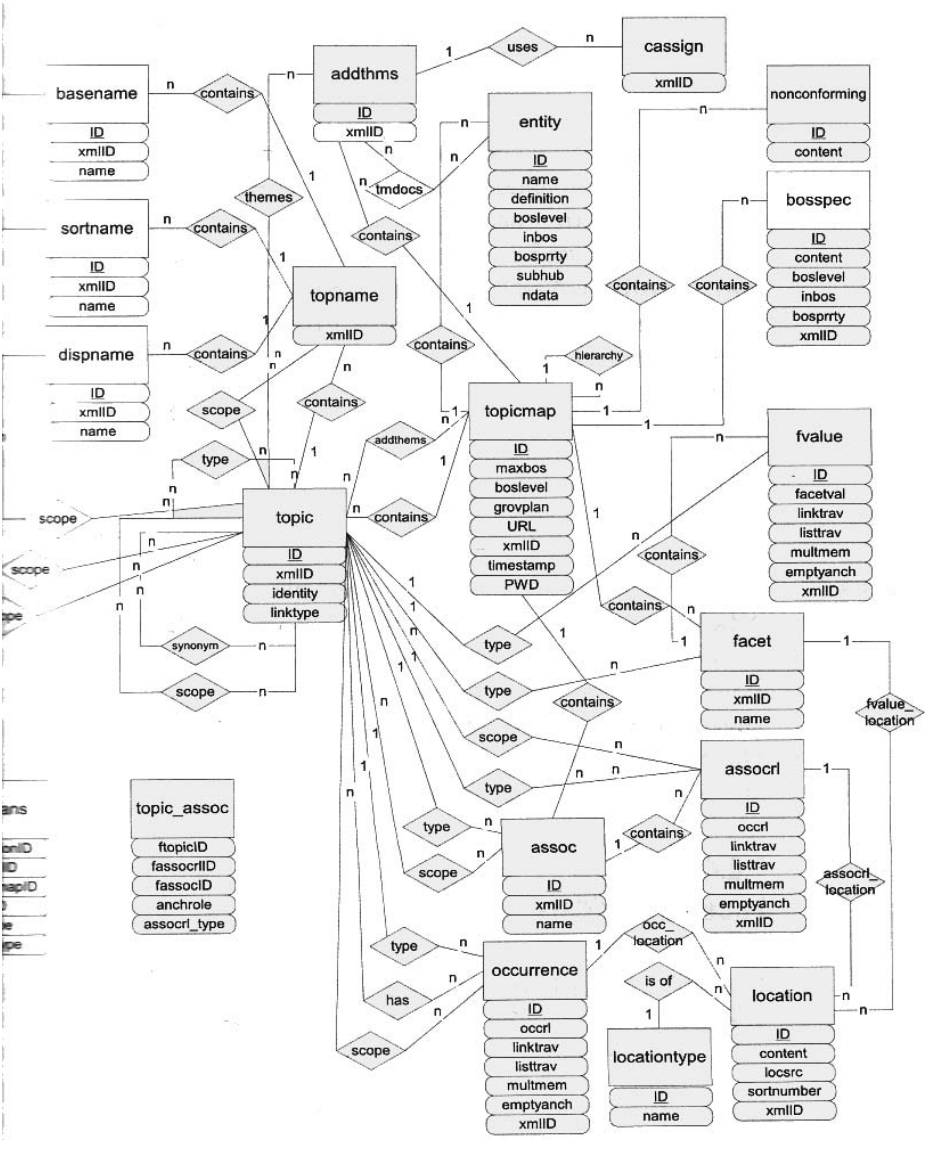

Fig. 4. Storing Topic Maps in an RDBMS [19]

3.4 Storing RDF

However, RDF can be stored similarly to Topic Maps by using either the "pure" link-based approach (Section 3.2) or modifying it in a way that is 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:

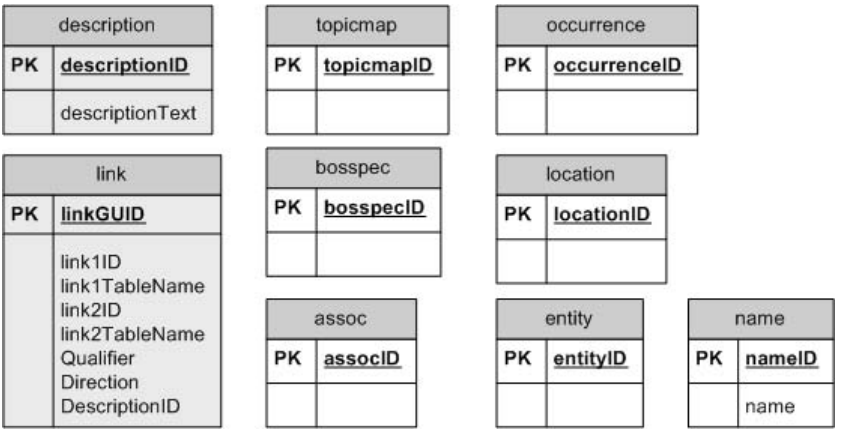

Fig. 5. By storing all relationships in the `link` table together with a qualifier, fewer tables (compare to Figure 4) are needed but all advantages as described in [19] are retained

1. In RDF, relationships can only be established between two resources whereas Topic Maps support relationships among 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 handled easily.
4. In RDF, reification is necessary if additional information must be attached to a relationship at a later time. This is not necessary for Topic Maps since everything is already reified. As shown previously, reification can be performed efficiently with our database schema.

Figure 6 shows how RDF data as described in [8] can be stored in our database structure. For efficiency and design considerations, we use five entities: Domain, Range, Resources, Property, and Class. All other entities described by [8] can be mapped by appropriate links and qualifiers in the `link` table.

Rather than using a table `subPropertyOf` (Figure 7) we qualify the recursive relation of property accordingly. `Literals` and `labels` are mapped to descriptions, the `type` to the qualifier of the `link` table and `namespaces` are implicitly defined in the description. `Range` is a qualifier of `domain`; `subClassOf` is mapped to `class` with a qualified recursive relation. The link table corresponds to the `triples`.

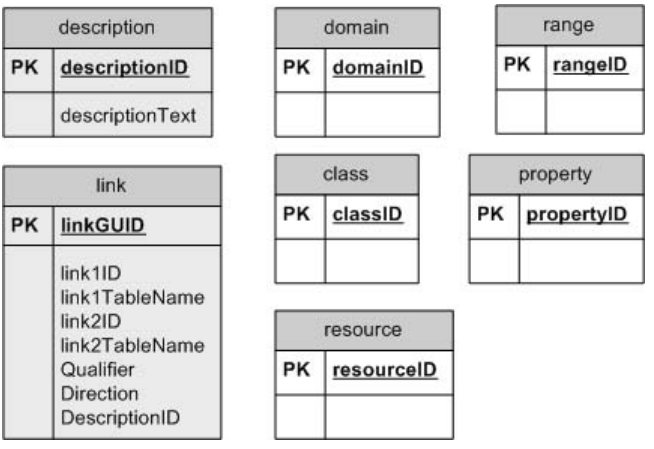

Fig. 6. Our database schema can store RDF (such as shown in Figure 7) independently of Topic Maps in the same schema

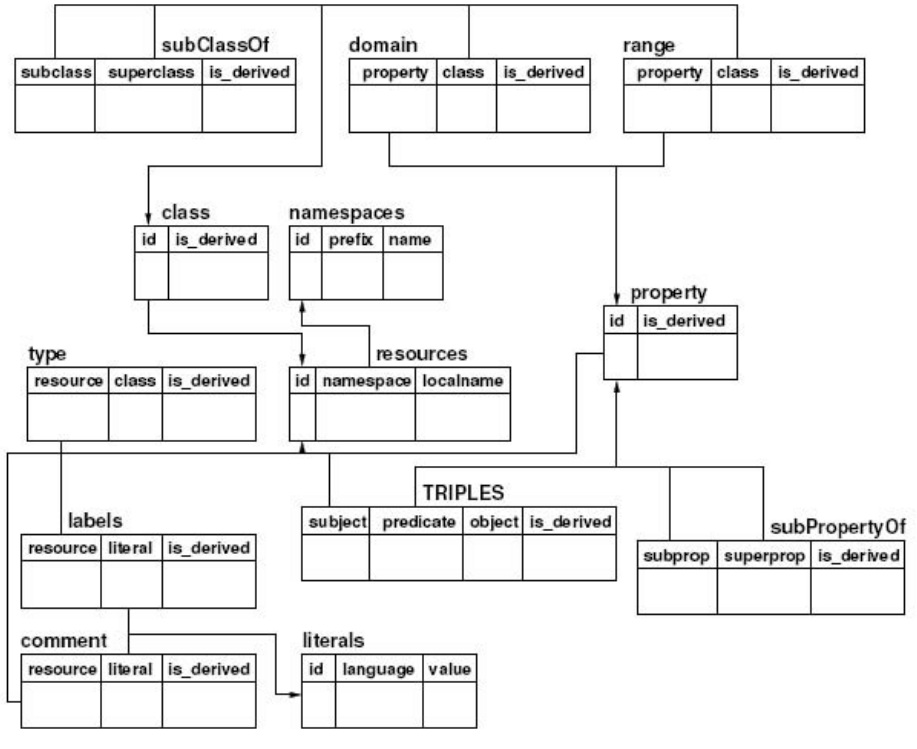

Fig. 7. The original RDF storage schema as proposed by [8]

4 Evaluation of Other Storage Concepts

In this section we look briefly at three systems that store personal information and strive to provide semantically enriched retrieval capabilities. For more details please refer to [18]. We then look at existing solutions (data-centric approach and structure-centric approach) to organizing a semantic data store.

4.1 Storing Personal Digital Information

Vanevar Bush's vision of the Memex [9]—a paper cited nearly universally when writing about semantically enriched information storage-provides the base for projects, such as, Microsoft's *MyLifeBits* [13] or the *SemanticLIFE* project [3] build. The authors aim to create a personal digital storage that records all of an individual's documents, emails, photos, videos, etc.

MyLifeBits focuses on storing digital content in a database; unlike SemanticLIFE its primary aim is not a semantic enrichment of the stored data. Instead, MyLifeBits relies on future improvement of search engines and desktop search solutions. The focus of SemanticLIFE is to build ontologies and discover relationships between existing data items.

Haystack [1] is a platform to visualize and maintain ontologies. The system is designed to flexibly define interactions and relationships between objects. 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 assumes that the MSSQL Server will provide all the needed functions without providing details on the database schema used.

4.2 Data-Centric Approach

One approach also known as a data-centric approach is often mentioned in the context of mapping XML documents to relational databases [15,5,6,16]. In terms of 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 corresponding 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 a situation in which 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 [7,8] in conjunction with a PostgreSQL database. Figure 8 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 instances 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 the data manipulation commands that are natively provided by all RDBMS.

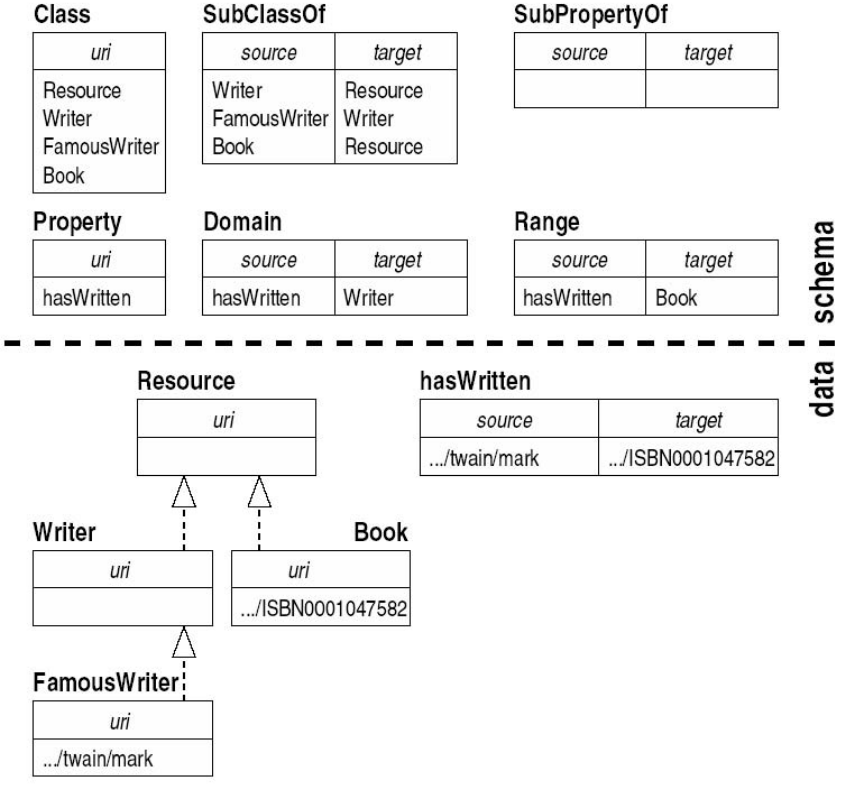

Fig. 8. Data-centric approach of Sesame [7]

Second, some RDBMS, such as, PostgreSQL, offer built-in object-relational features that can be used directly for modeling class-subclass relationships, etc. PostgreSQL databases offer, for instance, the possibility to create subtables that are connected to their parent tables through transitive relationships. This enables creating a table for a certain class and corresponding 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 are present. This means that changes to the class hierarchy always require the performance of data definition commands, which are expensive in almost any RDBMS.

4.3 Structure-Centric Approach

The second approach is also known as structure-centric and is equally popular among Topic Map and RDF implementations. As is the case with the data-centric approach, persistency is provided by a traditional RDBMS, but usually without requiring object-relational features. In contrast to the first approach, the key idea here is to map the finite number of data model concepts to corresponding structures (tables) in the relational database. Again, the process has also been described for XML documents [15,5,6,16], but additionally, has been specifically implemented for both Topic Map and RDF applications.

As shown in detail in Section 3.3, the Topic Map data model offers a small number of built-in concepts, such as, 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., [14,19].

With respect to RDF, the data model basically consists of statements only, 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 are available, mainly for achieving 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, it is not necessary to make any artificial distinction between 'schema layer' and 'data layer.' This not only allows for the representation of frequently changing ontology hierarchies, but also for efficient incremental incorporation of large datasets, since no structural changes of the underlying database schema are required.

The second advantage of structure-centric ontology representation is commonly reported for dedicated triple stores, but also applies to 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 does not directly address 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 that is expensive for large datasets [15,4]. Such large datasets must be seen as occurring frequently, as an ontology's entire information is stored within a single triple table. It is, therefore, common for such a table to contain millions of triples, and these triples must be compared to one another, often several times, depending on the nature of the query to be answered. Although various optimization efforts attempt to limit the negative effects of storing triples in a single table, in general, a lower level of performance in answering queries is to be expected as compared to the object-relational approach.

5 Conclusion

The Semantic Desktop Project aims at bringing the potential of RDF, Topic Maps, and Semantic Technologies to users' desktops. The goal is to develop a semantic personal information management system based on standards, such as, RDF, XTM and DAML+OIL/OWL, which assists users by automatically enriching collected data with semantic metadata.

Some important milestones are already in beta-testing, allowing performance tests and research regarding the querying of semantic statements. In this paper, we presented the current status of the project and proposed our improved method for storing ontologies in a relational database, which allows changes of hierarchies and relationships between tables to be added easily without schema modification.

The advantages of our approach are:

1. The modifications require no data-definition language (DDL) statements that 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 Topic Maps and RDF can be stored efficiently using our database schema.

Acknowledgements

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.

References

1. Adar, E., Karger, D., Stein, L.A.: Haystack: Per-user information environments. In: Proceedings of the Conference on Information and Knowledge Management (1999)
2. Agrawal, R., Somani, A., Xu, Y.: Storage and querying of e-commerce data. In: Proceedings of VLDB 2001, Rome, Italy (2001), http://www.vldb.org/conf/2001/P149.pdf
3. Ahmed, M., Hanh, H.H, Karim, S., Khusro, S., Lanzenberger, M., Latif, K., Elka, M., Mustofa, K., Tinh, N.H, Rauber, A., Schatten, A., Tho, N.M, Tjoa, A.M.: 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) (September 2004)
4. Alexaki, S., Christophides, V., Karvounarakis, G., Plexousakis, D.: On storing voluminous RDF descriptions: The case of web portal catalogs. In: ICSFORTH. Proceedings of the 4th International Workshop on the the Web and Databases (2001)
5. Bourret, R.: Xml-dbms, http://www.rpbourret.com/xmldbms/readme.htm
6. Bourret, R.: Mapping dtds to databases. Technical report, XML.com (2001), http://www.xml.com/lpt/a/2001/05/09/dtdtodbs.html
7. Broekstra, J., Kampman, A., van Harmelen, F.: Semantics for the WWW. In: Sesame: An Architecture for Storing and Querying RDF Data and Schema Information. MIT Press, Cambridge (2001), http://www.cs.vu.nl/~frankh/postscript/MIT01.pdf
8. Broekstra, J., Kampman, A., van Harmelen, F.: Sesame: A generic architecture for storing and querying rdf and rdf schema. In: Horrocks, I., Hendler, J. (eds.) ISWC 2002. LNCS, vol. 2342. Springer, Heidelberg (2002)
9. Bush, V.: As we may think. The Atlantic Monthly 176(7), 101–108 (1945)
10. Carroll, J.M., Rosson, M.B.: Paradox of the active user. ch. 5, pp. 80–111. Bradford Books/MIT Press (1987)
11. Dourish, P., Edwards, W.K., LaMarca, A., Lamping, J., Petersen, K., Salisbury, M., Terry, D.B., Thornton, J.: Extending document management systems with user-specific active properties. ACM Trans. Inf. Syst. 18(2), 140–170 (2000)
12. Ekelhart, A.: The blackman project: Collecting and querying semi-structured data for the 'semantic desktop'. Masterthesis, University of Technology Vienna, Vienna (2005)
13. Gemmel, J., Bell, G., Lueder, R., Drucker, S., Wong, C.: Mylifebits: Fulfilling the memex vision. In: ACM Multimedia '02, pp. 235–238. ACM Press, New York (2002)
14. Kiyakov, A.K., Simov IV, K., Dimitrov, M.: Ontomap: Ontologies for lexical semantics. Technical report, OntoText Lab, Sirma AI EOOD (2001), http://www.ontotext.com/publications/ranlp01.pdf
15. Kuckelberg, A., Krieger, R.: Efficient structure oriented storage of xml documents using ordbms. Technical report, RWTH Aachen (2003)
16. Mittermeier: Naiv nativ. iX 42(8) (2003)
17. Weippl, E.R., Klemen, M., Linnert, M., Fenz, S., Goluch, G., Tjoa, A M.: Semantic storage: A report on performance and flexibility. In: Andersen, K.V., Debenham, J., Wagner, R. (eds.) DEXA 2005. LNCS, vol. 3588, pp. 586–595. Springer, Heidelberg (2005)
18. Weippl, E.R., Klemen, M.D., Raffeiner, S.: The Semantic Web for Knowlege and Data Management: Technologies and Practices. In: Improving Storage Concepts for Semantic Models and Ontologies. Idea Group, USA (2007)
19. Widhalm, R., Mück, T.: Topic Maps: Semantische Suche im Internet. Springer, Heidelberg (2002)