3XL

An Efficient DBMS-based Triple-store

Xiufeng, Liu; Thomsen, Christian; Pedersen, Torben Bach

Published in:
The 23rd International Workshop on Database and Expert Systems Applications

DOI (link to publication from Publisher):
10.1109/dexa.2012.7

Publication date:
2012

Document Version
Early version, also known as pre-print

3XL: An Efficient DBMS-Based Triple-Store

Xiufeng Liu
Department of Computer Science
Aalborg University
Email: xiliu@cs.aau.dk

Christian Thomsen
Department of Computer Science
Aalborg University
Email: chr@cs.aau.dk

Torben Bach Pedersen
Department of Computer Science
Aalborg University
Email: tbp@cs.aau.dk

***Abstract*—This paper demonstrates the use of *3XL*, a DBMS-based triple-store for OWL Lite data. 3XL is characterized by its use of a database schema specialized for the data to represent. The specialized database schema uses object-relational features – particularly inheritance – and partitions the data such that it is fast to locate the needed data when it is queried. Further, the generated database schema is very intuitive and it is thus easy to integrate the OWL data with other kinds of data. 3XL offers performance comparable to the leading file-based triple-stores.**

We will demonstrate 1) how a specialized database schema is generated by 3XL based on an OWL ontology; 2) how triples are loaded, including how they pass through the 3XL system and how 3XL can be configured to fine-tune performance; and 3) how (simple and complex) queries can be expressed and how they are executed by 3XL.

I. Introduction

In recent years, the *Web Ontology Language*[1] (OWL), a semantic markup language recommended by W3C for publishing and sharing ontologies, has gained popularity. OWL is layered on top of the *Resource Description Framework*[1] (RDF). OWL (and RDF) data takes the form of (subject, predicate, object) triples. These triples are typically stored in specialized storage engines called *triple-stores*. We have seen that in some projects, the triple-stores are used mainly as specialized *bulk data stores*, i.e., for inserting and retrieving large amounts of triples (bulk operations). More advanced features such as logical inference, etc., are often not used in such projects. Additionally, for basic representation of OWL instances, we found that even a subset of the least expressive OWL layer (OWL Lite) was enough. A well-known example of such instances is the data generated by the data generator for the de-facto industry standard OWL benchmark Lehigh University Benchmark (LUBM) [7].

This paper demonstrates 3XL [11], a triple-store offering highly efficient loading and retrieval for OWL Lite data with a known ontology using a subset of OWL Lite. 3XL has a number of unique characteristics:

- 3XL is DMBS-based which makes it flexible and easy to integrate the OWL data with other data.
- Based on an OWL ontology, 3XL generates a specialized and intuitive database schema which intelligently partitions the data.
- 3XL uses object-relational features of the DBMS.
- Caching and bulk loading are intensively used in the implementation such that 3XL offers performance comparable to state-of-the-art file-based triple-stores.

This combination of efficiency and flexibility positions 3XL in a unique spot.

Fig. 1 shows the major components of 3XL and their interactions. 3XL consists of GUI and command line (CLI) interfaces, an API, a database schema generator, a data loader, a query engine, and an underlying DBMS (PostgreSQL). The database schema generator is responsible for parsing an OWL Lite ontology to create a hierarchical object-relational database schema according to a number of mapping rules. The data loader parses OWL data and inserts the data into the database. To speed up the data loading, it uses several cache and bulk schemes, including an embedded instance of BerkeleyDB. The query engine has a query parser and an executor which are used to generate SQL and run the SQL in the underlying database, respectively. In the paper, we show the GUI, but we note that a client application also could use the API.

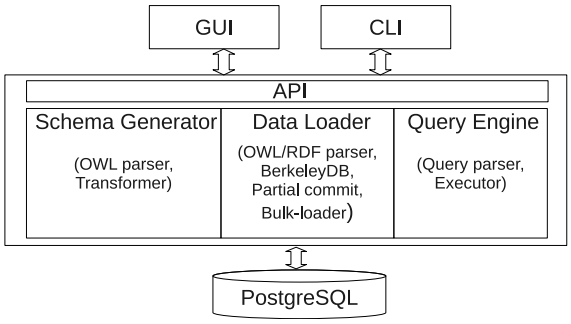

Fig. 1: 3XL Architecture

II. Specialized Schema Generation

Fig. 2 shows the interface for database schema generation. In this interface, the user can load and/or edit an OWL ontology. Further, the connection to PostgreSQL can be set up and the support for "multiproperty values" (to be explained later) can be configured. Based on the ontology (shown to the right in Fig. 2), a specialized database schema with an inheritance layout is generated once and for all by 3XL. We show how this is done by means of a running example.

Fig. 3 shows our small OWL Lite ontology which for ease and space reasons is inspired by the well-known LUBM

[1] www.w3.org/TR

ontology. For brevity, it is shown as a graph. The ontology defines classes (shown as ellipses), properties (shown as rectangles), and their relationships (shown as labelled edges). In our example, each student has exactly one email address and one name, and each course has exactly one name. A student can take several courses.

When creating a database schema, 3XL creates a *class table* for each class in the ontology. Class tables have columns for those properties that have owl:maxCardinality 1. Fig. 4 shows the resulting database schema for our example and it can, e.g., be seen that a class table is generated for Student. The Student class table has columns (name and email, both of type varchar) to represent the literal values of Student's data properties. When storing data, each Student instance results in a row in the Student class table.

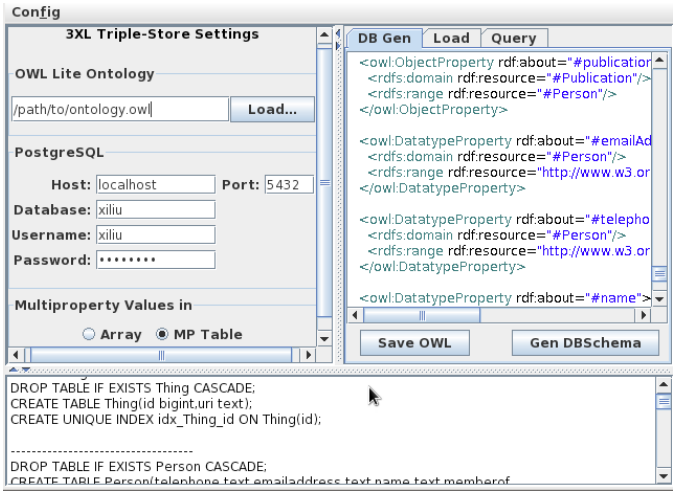

Fig. 2: GUI for database schema generation

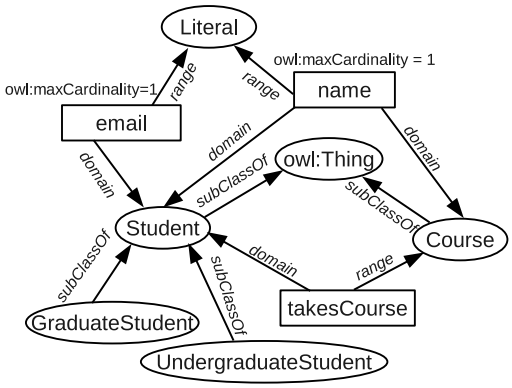

Fig. 3: OWL Lite schema

The class table for a given class *inherits* the class table for the parent of the represented class. In Fig. 4 it can be seen that the class tables for UndergraduateStudent and GraduateStudent inherit from the class table for Student. When reading from the Student class table, PostgreSQL also includes the data from the inheriting class tables. In OWL, every class is (explicitly or implicitly) a subclass of owl:Thing. Therefore, a class table representing owl:Thing is always present in a generated database schema. The class table for owl:Thing has two columns: ID (of type int) and URI (of type varchar) such that each of its descendants (i.e., each class table) at least has these two columns. The column URI represents the URI of the represented instance while ID represents a unique numerical identifier assigned by 3XL.

Data properties in the ontology result in columns of an appropriate type in the database schema (e.g., varchar for literal values as previously shown). For object properties, 3XL creates a column of type int. This column holds the IDs of the referenced objects and acts like a foreign key. However, the foreign key is not declared in the database schema and we thus denote the column as a *loose foreign key*. The reason for not declaring the foreign key is that it can reference IDs in more than one table if the referenced class has any subclasses.

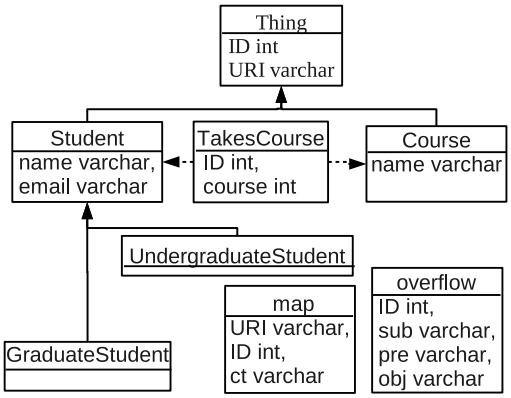

Fig. 4: 3XL database schema

In OWL Lite, some properties do not have maxCardinality 1. We refer to such properties as *multiproperties*. In the database schema, a multiproperty can be represented in two ways: By a column of an *array* type in the class table or by a special table called a *multiproperty table*. When an array type is used, it is possible to represent several values in a single column of a row. As can be seen in Fig. 2, it is possible to choose from the GUI how to represent multiproperties. In this example, we use multiproperty tables. As a student can take several courses, the takesCourse property is represented by means of a multiproperty table in Fig. 4. This multiproperty table holds a row for each value the multiproperty takes for a given instance. ID in the table for takesCourse is a loose foreign key to Student to represent which instance the value belongs to. As takesCourse also is an object property, a value of the property is represented by a loose foreign key to the class table for Course. If it had been a data property, a column of the appropriate type would have been used.

Based on the mapping rules [11] summarized in Table 1, 3XL generates DDL as shown in the bottom of Fig. 2. We emphasize how easy it is to understand and use a generated schema as the one in Fig. 4 with plain SQL (although queries by means of triples also are supported as demonstrated next).

This makes it very easy to integrate OWL data with other data.

TABLE I: Transformation rules

OWL constructs	Results in database schema
owl:Class	a class table
rdfs:subClassOf	inheritance between two class tables
owl:ObjectProperty or owl:DatatypeProperty	a column in a class table if owl:maxCardinality is 1, and in a multiproperty table or array column otherwise
rdfs:domain	a column in the class table if the maxCardinality is 1, and a loose foreign key from a multiproperty table to a class table otherwise
rdfs:range	a type for the column representing the property

Besides class tables and multiproperty tables, 3XL creates the table `map(URI, ID, ct)` which makes it fast to find the ID and/or class table representing an instance with a given URI. Although 3XL is specialized for the data with known ontologies, it can also store the triples that are not described by the ontology. It does so by creating the table `overflow(ID, sub, pre, obj)` which holds those triples.

III. Triple Loading

We demonstrate how data is loaded into 3XL and consider the following triples:

```
(http://www.univ0.edu/s0, rdf:type, Student)
(http://www.univ0.edu/s0, name, "s0")
(http://www.univ0.edu/s0, email, "s0@univ0.edu")
(http://www.univ0.edu/s0, takesCourse,
                    http://www.univ0.edu/course0)
(http://www.univ0.edu/course0, rdf:type, Course)
(http://www.univ0.edu/course0, name, "course0")
```

Data from triples with a common subject is first gathered in a *value holder* that maps from predicates to values. A value holder holds all known data about a specific individual. In the shown triples, there are two unique subjects. Thus, 3XL creates the two value holders shown in Fig. 5.

http://www.univ0.edu/s0	
ID	↦ *1*
rdf:type	↦ *Student*
name	↦ *s0*
email	↦ *s0@univ0.edu*
takesCourse	↦ *[2]*

http://www.univ0.edu/course0	
ID	↦ *2*
rdf:type	↦ *Course*
name	↦ *course0*

Fig. 5: Value holders

Each value holder also represents a unique ID assigned by 3XL. Further, the rdf:type is represented. In OWL Lite, the rdf:type must be explicitly given, but in many cases 3XL can also *deduce* the rdf:type based on the seen predicates. For example, only students have email addresses in our scenario. Multiproperty values, such as takesCourse, are represented as lists in value holders. Since takesCourse is an object property, the IDs of the referenced instances are held in the list instead of the (longer) URIs.

A value holder eventually results in a row in a class table and possibly some rows in multiproperty tables. The upper value holder in Fig. 5 results in a row in the class table for Student and a row in the multiproperty table for takesCourse. For efficiency reasons, 3XL only updates the underlying database in bulks. Thus, many value holders are held in a data buffer. When the data buffer is full, the value holders are transferred to the database in a bulk operation. This is shown in the upper part of Fig. 6.

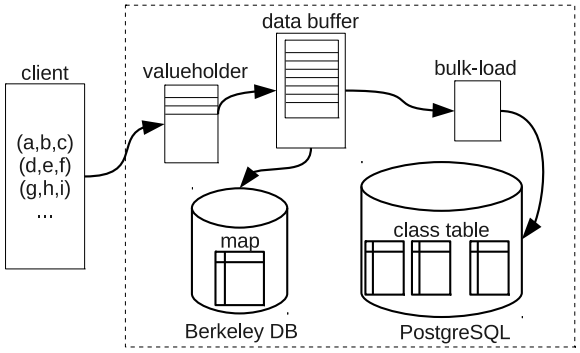

Fig. 6: Data flow in triple loading

When a value holder's data is inserted into the database, the value holder is deleted from the data buffer. When another triple is added, it can, however, happen that its corresponding value holder was just loaded into the database such that 3XL has to re-generate this value holder from the database. To avoid this, only the least-recently-used value holders are loaded into the database. We call this *partial commit*. This exploits that the data can have locality such that triples describing the same instance appear close to each other.

When the first triple in our example is seen by 3XL, there is no value holder for `http://www.univ0.edu/s0`. If a corresponding value holder for a new triple is not found in the data buffer, 3XL tries to look it up in the database. In our example, there is also no information about `http://www.univ0.edu/s0` in the database when 3XL sees the first triple. This is, however, difficult to determine as 3XL after seeing the first triple still does not know what type the instance has (and thus the appropriate class table is not known). Instead of searching all class tables, which would be very expensive, the `map` table is used. This table maps from a URI to the ID assigned by 3XL and the class table holding the data about the instance (i.e, from http://www.univ0.edu/s0 to the ID 1 and the class table Student in our case). For better performance, this table is loaded into BerkeleyDB which can keep a configurable amount of data in main memory. It is then very fast to determine which class table to look into or to decide that a new empty value holder should be created and no database lookup is necessary.

The full paper [11] details how bulk loading, partial commit,

and use of BerkeleyDB for `map` influence the performance.

3XL can also load the triples that are not described by the OWL Lite ontology used for the schema generation. If we, e.g., load the triple (http://www.univ0.edu/s0, `supervisor`, http://www.univ0.edu/prof0), the data will be represented in the `overflow` table.

Fig. 7 shows the interface for configuring the loading of triples into 3XL. Triples in the N3-format can be loaded from a file using this interface. It is possible to configure the amount of memory used by BerkeleyDB. Further, the size of the data buffer can be set as well as the percentage of least-recently-used value holders to insert into the database when the data buffer is full. It is thus possible to fine-tune 3XL's performance. Details about the configuration parameters are available in [11] and will be discussed during the demonstration. With proper configuration, 3XL achieves load-speeds of up to ∼25,000 triples/second on a normal notebook [11] which is comparable to the leading file-based triple-stores BigOWLIM [2] and RDF-3X [12] as explained in Section V.

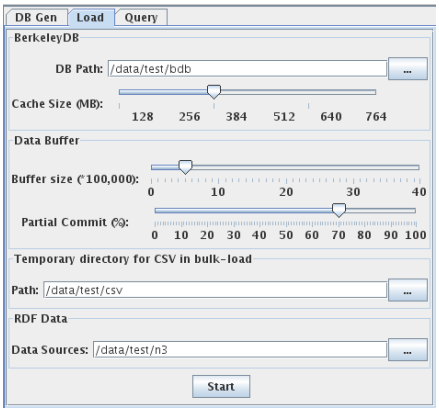

Fig. 7: 3XL's GUI for triple loading

IV. Triple Queries

We now demonstrate how the data can be queried. Fig. 8 shows the interface for querying. A query can be entered in the upper part of the window and the result can either be written to a file or shown in the lower part of the window. 3XL supports two classes of queries: point-wise and composite queries. They are described in the following.

Point-wise Queries: A *point-wise query* is a single triple, i.e., $Q = (s, p, o)$. The result includes all triples that have identical values for s, p, and o. Each part of the query triple may, however, be a wildcard "*" which is considered to be identical to everything. If we issue the query `(http://www.univ0.edu/s0, takesCourse, *)` on the previously loaded triples, the result is (http://www.univ0.edu/s0, `takesCourse`, http://www.univ0.edu/course0).

To answer such queries, 3XL first identifies the relevant class tables and the IDs of the instances by means of the `map` table. In our example, 3XL thus finds the ID 1 for the instance. The class table for Student does, however, not hold

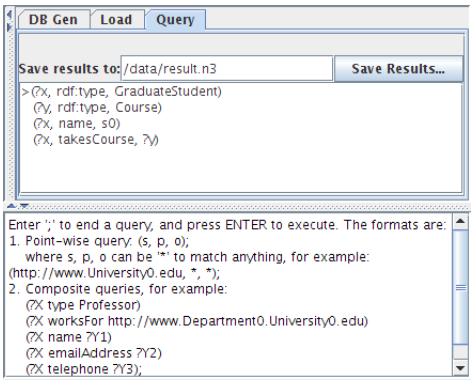

Fig. 8: 3XL's GUI for triple querying

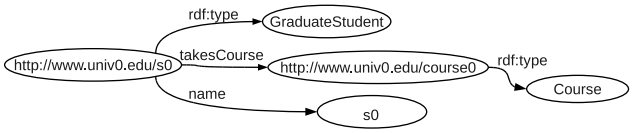

Fig. 9: The result of the composite query

the information needed to answer the query. The reason is that takesCourse is a multiproperty and we use multiproperty tables. As takesCourse also is an object property, 3XL has to join with the class table of takesCourse's range (i.e., Course) to find the URIs of the courses taken by the represented student:

```
SELECT Course.URI FROM TakesCourse, Course
WHERE TakesCourse.courseId = Course.ID AND
      TakesCourse.studentID = 1
```

From the result of this SQL query, 3XL generates the triple result set – which in this case consists of a single triple. For the query `(http://www.univ0.edu/s0, email, *)` it is enough to identify the relevant class table (by means of the `map` table) and then select the email address from that class table. The reason is that `email` has maxCardinality 1 and is a data property. These examples illustrate how 3XL efficiently answers the important category of $(s, p, *)$ queries. For details about these and other point-wise queries, see [11].

It should be noted that when a query is made on a table with inheriting tables, the inheriting tables are queried as well. If, for example, a query is made on the Student table, the UndergraduateStudent and GraduateStudent tables are also queried due to PostgreSQL's object-relational features.

Composite Queries: 3XL also supports *composite queries*. Composite queries consist of several query triples and are more expressive than point-wise queries. Unknown *variables* used for *linking* triples together are specified using a string starting with a question mark while known *constants* are expressed using their full URIs or in an abbreviated form where prefixes can be replaced with shorter predefined values. The upper part of Fig. 8 shows an example of a composite query to find the courses taken by a student with a certain name. Fig. 9 shows the result as a graph. The result consists of all triples as those in the query, but with variable names replaced by actual values.

To answer a composite query, 3XL generates more complex SQL than for the point-wise queries. For the shown example, the following SQL is generated. 3XL can then generate the triples of the result.

```
SELECT GraduateStudent.uri AS x, Course.uri AS y
FROM GraduateStudent, TakesCourse, Course
WHERE GraduateStudent.ID = TakesCourse.ID AND
     TakesCourse.takesCourse = Course.id AND
     GraduateStudent.name = 's0'
```

V. Performance

We study the performance of 3XL by comparing with the two leading file-based triple-stores BigOWLIM [2] and RDF-3X [12] and use a synthetic data set (LUBM [7]) and a real-world data set (EIAO [14]), both with 100 million triples. With proper configuration on a normal notebook, 3XL loads 18,519 triples/s (EIAO) and 24,765 triples/s (LUBM) and outperforms RDF-3X which loads 10,502 triples/s (EIAO) and 10,704 triples/s (LUBM). BigOWLIM loads 19,267 triples/s (EIAO) and 30,030 triples/s (LUBM). The query performance of 3XL is also comparable to BigOWLIM and RDF-3X on the EIAO data set (3XL is best for 4 of 10 queries) as well as the LUBM data set (3XL is best for 6 of 14 queries). Generally, file-based triple-stores are considered to be faster than DBMS-based ones. It is thus remarkable that 3XL offers the both of best worlds: performance *and* flexibility. For more details about the performance study, see [11].

VI. Related Work

There are many different RDF storage systems available. Haslhofer et al. conduct a survey of the existing RDF stores and categorize and compare the existing solutions [10]. As RDF data consists of *(subject, predicate, object)* triples, many early systems, including 3store [8] and GridVine [5], use a giant and narrow triple table to represent all the triples. This approach is different from the approach taken by 3XL where the data to store is held in many class tables. Hexastore [15] and RDF-3X [12] optimizes RDF query processing by building indices according to the permutation of the columns of the triple table. They both do not use a DBMS, but their own database engines. Das et al. [6] compare file-based and DBMS-based triple-stores and show that the file-based ones are generally faster. Various approaches are proposed to speed up query processing by considering clustering RDF data based on properties, including [1], [4], and [16]. They are orthogonal to 3XL. 3XL, however, is specifically designed to be fast for queries where the subject and/or predicate is known and where many/all properties should be retrieved. In addition, a number of industry-oriented solutions exist, such as [2], [3], [6], and [9]. They use system files or a DBMS as the back-ends for storing data. 3XL uses an object-relational DBMS as the back-end, but takes advantage of advanced features such as inheritance. This is is an intuitive way to represent RDF/OWL data in the ORDBMS and is both easy and fast to use.

VII. Demonstration

In the demonstration of 3XL, we use a LUBM-inspired ontology to generate a specialized database schema. The ontology covers all classes and properties used by the LUBM data generator. First, we show how the ontology is mapped into a database schema including how multiproperties are handled and how to handle triples describing unknown classes. Second, we demonstrate data loading by using the LUBM data set. We demonstrate how triples are inserted and how the 3XL system processes them on their way into the system. We further show the performance effects of the sizes of the cache and partial commits. Third, we show querying on the loaded data, including both point-wise and composite queries. Finally, we demonstrate how the data can be integrated with other (non-triple) data. The audience will see how one can interact with 3XL through the GUI client and interactively generate database schemas, load data, and execute queries.

References

[1] D.J. Abadi, A. Marcus, S.R. Madden, and K. Hollenbach. Scalable Semantic Web Data Management Using Vertical Partitioning. In *Proc. of VLDB*, pp. 411–422, 2007.

[2] BigOWLIM - Semantic Repository for RDF(S) and OWL. owlim.ontotext.com/display/OWLIMv35 as of 2012-05-01.

[3] J. Broekstra, A. Kampman, and F. van Harmelen. Sesame: A Generic Architecture for Storing and Querying RDF and RDF Schema. In *Proc. of ISWC*, pp. 54–68, 2002.

[4] E. I. Chong, S. Das, G. Eadon, and J. Srinivasan. An Ecient SQL-based RDF Querying Scheme. In *Proc. of VLDB*, pp 1216–1227, 2005.

[5] P. Cudre-Mauroux, S. Agarwal, and K. Aberer. GridVine: An Infrastructure for Peer Information Management. *Internet Computing, IEEE*, 11(5):36–44, 2007.

[6] S. Das, E. Chong, W. Zhe, M. Annamalai, and J. Srinivasan. A Scalable Scheme for Bulk Loading Large RDF Graphs into Oracle. In *Proc. of ICDE*, pp. 1297–1306, 2008.

[7] Y. Guo, J. Heflin, and Z. Pan. LUBM: A Benchmark for OWL Knowledge Base Systems. *Web Sem.*, 3(2):158-182, 2005

[8] S. Harris and N. Gibbins. 3Store: Efficient Bulk RDF Storage. In *Proc. of PSSS*, pp. 1–15, 2003.

[9] S. Harris, N. Lamb, and N. Shadbolt. 4store: The Design and Implementation of a Clustered RDF Store. In *Proc. of SSWS*, 2009.

[10] B. Haslhofer, E. M. Roochi, B. Schandl, and S. Zander. Europeana RDF Store Report. In University of Vienna, Technical Report, 2011. eprints.cs.univie.ac.at/2833/1/europeana_ts_report.pdf as of 2012-05-01.

[11] X. Liu, C. Thomsen, and T. B. Pedersen 3XL: Supporting Efficient Operations on Very Large OWL Lite Triple-stores. *Inf. Sys.*, 36(4):765–781, 2011

[12] T. Neumann and G. Weikum. RDF-3X: A RISC-style Engine for RDF. *PVLDB*, 1(1):647–659, 2008

[13] OWL Web Ontology Language Guide. www.w3.org/TR/owl-guide as of 2012-05-01

[14] C. Thomsen and T.B. Pedersen. Building a Web Warehouse for Accessibility Data. In *Proc. of DOLAP*, pp. 43–50, 2006.

[15] C. Weiss, P. Karras, and A. Bernstein. Hexastore: Sextuple Indexing for Semantic Web Data Management. *PVLDB*, 1(1):1008-1019, 2008.

[16] K. Wilkinson, C. Sayers, H. Kuno, and D. Reynolds. Efficient RDF Storage and Retrieval in Jena2. In *Proc. of SWDB*, pp. 131–150, 2003.