Semantic Storage: Overview and Assessment

Alisdair Owens
University of Southampton
awo101@ecs.soton.ac.uk

ABSTRACT

The Semantic Web has a great deal of momentum behind it. The promise of a 'better web', where information is given well defined meaning and computers are better able to work with it has captured the imagination of a significant number of people, particularly in academia. Language standards such as RDF and OWL have appeared with remarkable speed, and development continues apace. To back up this development, there is a requirement for 'semantic databases', where this data can be conveniently stored, operated upon, and retrieved. These already exist in the form of triple stores, but do not yet fulfil all the requirements that may be made of them, particularly in the area of performing inference using OWL. This paper analyses the current stores along with forthcoming technology, and finds that it is unlikely that a combination of speed, scalability, and complex inferencing will be practical in the immediate future. It concludes by suggesting alternative development routes.

KEYWORDS

triple, store, RDF, RDFS, OWL, semantic, inference

1. Introduction

The Semantic Web claims to offer us a 'better web'; a web where information is given meaning that computers can identify, allowing them to work with that information in a more intelligent manner [2]. Semantic Web technologies already see a significant degree of use, particularly in the world of academia.

While several of the core language technologies have already been created, less attention has been paid to how the information these languages encode will actually be stored. This paper begins with a background explanation of the information that that has to be stored, and goes on to examine the present and future of semantic storage systems, with particular reference to speed, scalability and inferencing capability they can provide.

2. Languages

The existing World Wide Web is comprised largely of data designed for consumption by human beings. The data can be duplicated or contain contradictions, and there is no formal style required for data presentation. There are no guarantees made that information will always be accessible. This makes the web an extremely difficult medium for machines to navigate, but the information on the web is easy for humans to understand and navigate with the aid of tools such as search engines. A lack of guarantees on both web connectivity and formal presentation has allowed the web to grow at an impressive rate.

The creators of the Semantic Web hold a vision of a World Wide Web that contains data that can be 'understood' by machines. Once machines can assign a degree of meaning to data and effectively traverse it, great possibilities for both task automation and data gathering can be realized. Required for the Semantic Web, then, are technologies that can express information, express rules, and use rules to perform inference and answer questions. Some of these technologies are already in place, and are considered below.

2.1 RDF

The Semantic Web community's solution to adding meaning to data is the Resource Description Framework. RDF encodes meaning as subject, predicate, object triples, and is written using XML.

An example of this might be (Roy Owens, has-son, Alisdair Owens). This would simply encode the information stating who I am the son of. If you added the triple (Alisdair Owens, has-university, University of Southampton), one could reasonably determine that the son of Roy Owens attends Southampton University. This structure is a simple way to describe most data that machines are likely to process.

RDF is a graph data structure. Each triple represents an arc in the graph. Written in triple form, RDF has no requirement for ordering thanks to the fact that the 'subject' and 'object' are fully identified in each triple. RDF defines three containers that can be used to group information together: Bags, Sequences, and Alts. Bags and Alts are unordered containers, but in sequence the order in which the statements are read is significant.

'Triple Stores' refer to storage mechanisms originally designed to hold the data in RDF triples.

2.2 RDFS

RDF Schema is RDF's vocabulary description language. It is a semantic extension of RDF used to describe groups of related resources and the relationships between those resources. RDFS is used to define class and property systems similar to those you might find in object oriented programming languages.

Most current semantic storage systems are capable of taking advantage of the basic level of inference provided by RDFS, but few are currently capable of working with the next technological in the Semantic Web 'layer cake' – ontologies.

2.3 Ontologies

An ontology is a formal specification of a knowledge domain: what classes of data there is in that domain, the relationships between these classes, and the manner in which they are formed.

The Web Ontology Language (OWL) is used to share knowledge in such a manner that a variety of machines can understand it; using an ontology one can define data in such a manner that it will 'mean' the same thing to different machines. Once a knowledge domain is described using OWL, one can use a machine to perform reasoning upon data (for example, data defined in RDF).

This has a tremendous amount of utility, in that new information can be inferred from existing data. For example, consider figure 1 below:

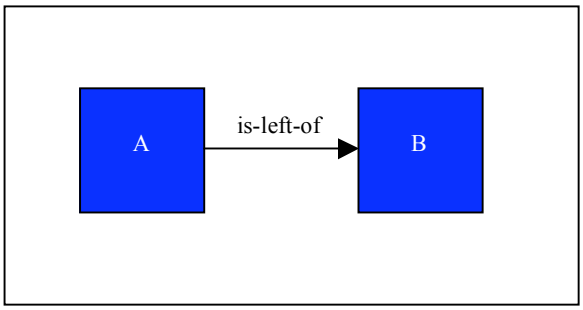

Figure 1.

In this scenario, all we know is that box A `is-left-of` box B. If, using an ontology, we were to state that the property `is-right-of` was the inverse of `is-left-of`, we could then infer that box B `is-right-of` box A.

OWL provides three increasingly expressive and complex sublanguages: Lite, DL, and Full. OWL Lite is for those who require a classification hierarchy and simple constraints. OWL DL supports all OWL language constructs, but limits the manner in which they are used to retain computational completeness and decidability. OWL Full allows the freeform expressiveness of RDF, but does not guarantee that all possible inferences will be computable or that they will finish in a finite time.

2.4 Triple Stores

While languages to encode semantic information are already available, the information must also be stored in an effective manner. Triple stores are the database of the Semantic Web world, designed to hold massive numbers of RDF triples in such a manner that the information they encode can be simply retrieved. Triples stores can already contain a vast amount of information; 3Store, perhaps the fastest and most scaleable store currently in existence, has been observed holding over 40 million triples [8]. While this is sufficient for most applications, common relational database solutions can effectively scale to hundreds of gigabytes or terabytes of data. Owners of large database-backed web sites that they wish to semantically enable, for example, may find that current triple stores lack the scalability to support their needs.

As well as scalability, response times (for the purposes of this report, 'performance') are a significant issue. Many existing Semantic Web applications interact with humans, and to prevent the human getting frustrated the application must perform as fast as possible. A response time suitable for human interaction is referred to as 'interactive level performance'.

A final clear issue is support for inferencing. While RDFS support is common, only a very few stores support OWL features, and they do not offer the level of performance when utilizing these features that the simpler systems provide.

2.4.1 Querying

While the most basic triple stores only allow the extraction of one triple at a time, those intended for serious use usually have an ability to respond to complex queries. While there are several competing querying implementations, the de facto standard for the current generation of stores is the RDF Data Query Language (RDQL). RDQL, like most other query languages of its kind, matches a chain of triple patterns against the RDF graph. These triple patterns can contain variables, allowing for the possibility of multiple matches. Once the store has determined all matches, it returns them. In the next generation of stores, RDQL is likely to be succeeded by SPARQL, which operates in a very similar manner.

3. Literature Investigation

This section summarises the ten papers selected for investigation into semantic storage mechanisms and their future.

3.1 Sesame [4]

Paper [4] describes the creation of Sesame, an early triple store that is still under continuous development. It is designed to provide RDF querying capabilities independently of the backing store upon which it is based. After summarising the RDF and RDFS language technologies in a similar manner to that seen in section 2, the paper goes on to describe Sesame as a store designed to provide RDF querying at the *semantic* level, that is, it provides graph pattern matching.

As seen in figure 2, Sesame layers metadata capabilities on top of existing storage systems, using its Repository Abstraction Layer to provide a constant interface with the data repository to its other components. These storage systems, while indeterminate, affect the performance of the program as a whole. While flat file storage is possible and offers increased speed for small, import heavy applications, relational databases such as MySQL and PostgreSQL are recommended. Relational database backing ensures that the repository itself will scale effectively, and be able to respond to data requests within an acceptable period of time. This is essential for querying performance.

This architecture necessitates that the querying take no advantage of repository-specific shortcuts to improve querying speeds. The fact of in-memory querying means that RQL queries on moderately sized data stores may take 60ms or more, which is potentially too slow for many applications. Currently, the query module supports RDF/S, and has extensions that provide some OWL Lite features.

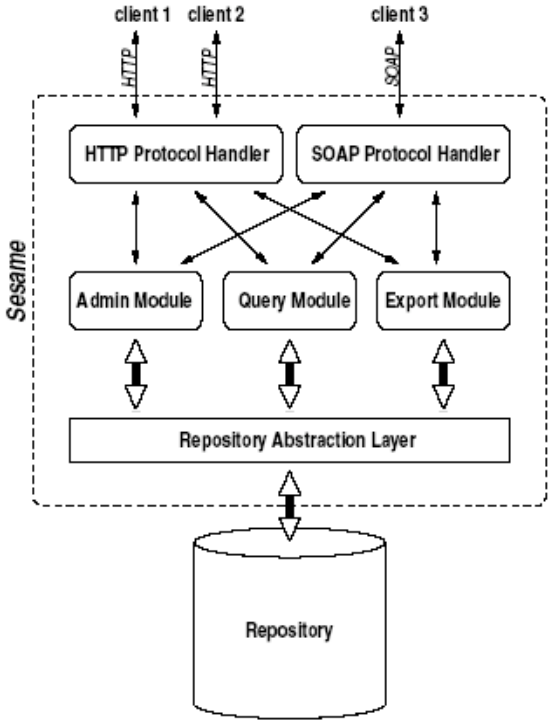

Figure 2. Sesame architecture

3.2 3Store [8]

This paper describes 3Store, a triple store system designed to improve on the scalability and performance limitations seen in systems like Sesame and Jena at the time of creation. It takes the form of C libraries operating on top of a MySQL storage layer. The focus for this project is the ability to scale to large data stores, and it is known to be capable of working with over forty million triples.

The paper notes that performance considerations for 3Store are separated into two categories: querying performance and the time to assert new data. Many applications require interactive-level performance from querying, particularly in web applications that users will expect to be responsive in short time constraints. This need for querying performance is satisfied by the use of a single target backend of MySQL. Given a known backend, rather than perform queries in memory, 3Store translates RDQL into optimised SQL queries. Querying in the storage layer provides noticeable performance gains. Data assertion is also aided by database-specific optimisations, and a data set of several million triples can be asserted in a few hours.

triples

model	subject	predicate	object	literal	inferred
int64	int64	int64	int64	boolean	boolean

models

hash	model
int64	text

resources

hash	uri
int64	text

literals

hash	literal
int64	text

Figure 3. 3Store schema

Figure 3 describes the database schema used by 3Store, and shows it to be a 'traditional' RDF store, in that it uses a few very large tables.

On the topic of inference, the paper differentiates between purely 'forward chaining' systems like Sesame that precompute all possible inferences on assertion and add them into their data store, and backward-chaining applications that produce inferences dynamically from queries presented to the system. It states that while forward chaining produces significant performance benefits, it can lead to an excessive increase in the amount of data stored in the system. Conversely, backward chaining is space-efficient, but can cause the query process to proceed intolerably slowly for interactive-level applications. 3Store adopts a hybrid process, where RDFS entailments that do not generate an excessive amount of extra data are produced by forward-chaining upon assertion, while others are produced by backward chaining from queries. Inference in 3Store is limited to the RDFS level. The creator of 3Store expects that implementing any of the OWL species would result in an intolerable loss of performance.

3.3 RDFStore [3]

RDFStore is a C/Perl toolkit designed to allow the storing of RDF triples. This paper begins by identifying the difficulties inherent in storing RDF data in a traditional relational database: RDF allows data to be stored without constraints upon the relationships it holds to other data, allowing deep nesting and even cyclic relationships. RDBMS are capable of storing this kind of data, but tend to process it inefficiently. A lack of predefined structure makes it extremely difficult to optimise retrieval.

The paper notes that most existing triple stores function on top of relational databases, and have to force RDF data into a few tables with many rows, resulting in a large number of database queries to satisfy a single RDF query. The paper's presented solution to this problem is the use of Berkeley DB, a database system that has no notion of the schema specification that is so fundamental in relational databases. Fundamental to allowing generic querying in this architecture is a hash table for each 'subject' of an RDF triple containing all possible connections of a resource node present as subject in a certain statement to all the other statements where this resource node is present as an object or predicate. The same principle is applied to hash maps of predicates and objects. Using these hash tables, it is relatively fast to perform pattern matches on sub queries.

3.4 Kowari [1]

Kowari might be considered something of a leap beyond previous triple store systems in that it uses a storage system designed specifically for the purpose, rather than relying on a flat file or database layer below it. The paper notes that 'triples' in Kowari are actually quads; an extra data item is added to indicate provenance. This approach is seen in other triple stores, but they do not usually fully index the provenance information. Again, as in other triple store systems, URIs and literals are referred to as 64 bit integers. These are stored in the 'Node Pool', while the URIs and literals themselves are stored in a 'String Pool' that stores the physical strings.

The String Pool uses AVL trees to store indexes into files which hold the RDF data in its integer indexed form, with a corresponding reverse node to relate integer data to string form. The statement indexes use 6 parallel AVL trees to index all the required orderings of subject, predicate, object, and meta

(provenance) node. The use of AVL trees makes satisfying queries a relatively simple matter, in that the triples can be rotated to produce matches against the index trees.

A result of this application-specific implementation is excellent scalability. Running on a 32 bit processor, Kowari has been shown to load 50 million triples before performance starts to degrade, while use of a 64 bit processor increased this amount to 170 million triples.

Kowari is unusual amongst large scale triple stores in that it implements some basic OWL functionality, in the form of OWL Lite, and the developers expect implementations of both OWL DL and OWL Full in the future. As in 3Store, rules which generate relatively few extra triples are produced by forward chaining from asserted data, while the others are produced by backward chaining from queries.

3.5 Jena2 [10]

Jena was an early RDF storage system designed to run as local storage rather than as a remote server. Jena2 is the replacement, designed with a focus upon improving performance and scalability. Jena offers the ability to store data in indeterminate backing stores, but the most common backing is a relational database. As seen in most other triple stores, Jena 1 used a simple database schema of one table for statements, one for URIs, and one for literals. This design results in the requirement for multiple join operations to run a FIND.

Jena 2 improves performance by denormalising the schema; simply storing URI/literal information in the same table row as the statements. This increases storage requirements, but improves the speed with which data can be found. Common namespaces and large literals are also stored in separate tables.

3.6 Existing Store Performance [11]

This paper describes a study of various performance features of semantic data stores, and appears to be the most comprehensive of its type currently available. The authors ran an RDF browsing application called Longwell on top of various kinds of triple store, noting in particular the time required for assertions and store initialisation, and the time taken to display the first page of the application, which is reliant on time taken for RDQL queries. The experiments test the Kowari, 3Store, Jena, Sesame, and Joseki stores, with a particular focus on stores acting as remote servers rather than local models.

The conclusions from the tests show that 3Store and Sesame offer the overall best performance in terms of configuring data, with Kowari a short distance behind. Unfortunately, the results for testing query performance do not show significant differences between the stores, despite this being noticeable in practical use. The paper notes that while file-based systems offer the fastest import times, that database backed systems (in particular MySQL 3) provide superior scalability.

It should be noted that there are significant issues in the test methodology that make the relevance of some of the results questionable. Firstly, the data set is relatively small at fewer than 300,000 triples, and is highly disconnected. A small, flat graph structure is unlikely to tax querying engines, which may be an explanation for the lack of disparity in querying results. Finally, each of the stores has only one connection made to it at a time, so no information is available on how the performance alters under stress.

3.7 Large RDF Datasets [7]

Application-Specific Schema Design for Storing Large RDF Datasets [7] begins with an affirmation of the limitations of current triple store architectures as described in [3]; that is, the data is forced into a few extremely large tables, and is thus not tractable for optimisation thanks to a lack of data-specific schema specification.

The paper notes that while RDF is unstructured, and it is thus difficult to determine more specific schemas for it, complex schemas can be created if we have prior knowledge of how the RDF is to be structured. That is, if we knew that all the data in our store was going to be about *people*, and each *person* might potentially have data in the set [Forename, Surname, Address, Telephone number], one might simply construct a more specific database schema to hold the data and permit greater optimisation.

While this kind of optimisation is useful, it does not cater for the realistic likelihood that the data being entered into the store is an unknown quantity. The paper goes on to describe an optimisation system, involving iterative improvement of schema design using on a schema generation and testing loop, based on discovering patterns in the RDF data and the queries being used to extract information from it. The initial set of data is altered to fit the new schema by an automatic data generation component, as well as being expanded upon to ensure that there is sufficient data for testing.

Currently, the work is limited to static RDF data, and addition of new data would require reanalysis. Clearly, however, application-specific schemas offer significant potential for scalability and performance increases. The creator of 3Store believes that such work might enable stores to hold up to 10^9 triples effectively.

3.8 Peer to Peer Systems [5]

This paper highlights work into creating peer to peer RDF storage systems. This is of particular import, given that the vast majority of existing triple store systems are self contained; a single application running on top of a single database. Distributed stores offer the potential for greatly increased scalability with thanks to the power of multiple machines doing the work. Such work may also constitute the start of development on ways to effectively run queries over data on multiple different individual stores, which is a large unsolved problem in the Semantic Web world as it stands.

RDFPeers builds upon an existing self organising peer to peer system called a Multi Attribute Addressable Network (MAAN), and adds RDF storage/retrieval capabilities on top of that. The information for each RDF triple is stored at three locations within the network, the choice of location being based upon a SHA1 hash of each of the subject, predicate, and object. Standard peer-to-peer functionality such as replicating information on neighbouring nodes in case of failure is implemented. Querying is built on top of this architecture, including complex conjunctive multi-predicate queries. However, while atomic queries can be resolved in an O(log N) time period (where N is the number of routing hops per query), conjunctive queries are still rather expensive.

3.9 Inferencing [6]

This paper describes the implementation of inferencing capabilities in the Jena RDF store. Jena currently provides a moderately sophisticated inferencing engine, with full RDFS

support and a feature set approximately equivalent to that of OWL Lite.

As described in section 3.2, there is a choice to be made between forward chaining, backward chaining, and a hybrid of the two. Forward chaining sends the entire data set to the rule engine, notes any deductions that fire off, and adds them to the data set. Backward chaining uses a logic programming engine similar to Prolog engines. When the data model is queried the query is translated into a goal and the engine attempts to satisfy that goal by matching against any stored triples and by goal resolution against the backward chaining rules. Hybrid engines perform forward chaining upon rules which do not generate much data, and rely on backward chaining for the rest.

Forward chaining generates a significant amount of extra data, and the need to check that inferences do not affect the rest of the data model when new data is added means that as the data grows the cost of adding data increases. Backward chaining has a significant cost upon querying, and is rarely suitable for interactive-level performance.

3.10 Inferencing Performance [9]

All of the mentioned stores except for RDFPeers include RDFS support, but only Jena, Kowari and Sesame (with a custom extension) offer OWL features. Even these more advanced reasoners are limited to the relatively simple OWL Lite. This paper [9] considers the performance implications of implementing reasoners. This work was accomplished through the creation of a benchmark that asserts a relatively small ontology with a significant (between 10^5 and $6*10^6$ triples) amount of data, and performs queries to extract information from this asserted data. The paper considers the performance, soundness, and completeness of these responses.

Initially, testing was intended to be performed on a wider variety of OWL reasoners including Jena, and Racer. Unfortunately, Jena's performance became unacceptable with reasoning turned on, and Racer was incapable of scaling to the dataset. Remaining were four platforms: OWLJessKB, Sesame in-memory, Sesame-database, and DLDB-OWL. The tests revealed that OWLJessKB was unable to scale beyond the smallest datasets, while Sesame's forward chaining reasoner failed on the largest datasets. DLDB (which uses a backward chaining reasoner) was capable of asserting the large data sets. Query timing shows that DLDB's response times for large datasets and/or more complex problems, would be in the range of several seconds, which is entirely unsuitable for interactive-level performance. Query response time appeared to scale in a linear fashion with the amount of data in the store. It should be noted that Sesame was only tested with RDFS support, and indications from the creators are that the OWL Lite extension will not scale at all effectively.

4. Research Methodology

Research for this paper was largely undertaken through the use of Google Scholar, IEEE eXplore, and the ACM digital library. Where a paper was found to be of interest, I followed potentially useful references. Finally, Stephen Harris, the creator of 3Store, was a great deal of help in answering questions and pointing me towards useful information.

5. Project Outcome

As described in the introduction, there are three obvious problems to be solved: speed, scalability, and inferencing capability. Further research has revealed another issue somewhat tied in with scalability: the problem of distributing queries across multiple stores. This is important so that we can distribute data, both for reasons of performance and simply because we will not all wish to store our data on one centralised server.

The direction in which we should be heading depends upon which of these problems is most important to us. Some applications will demand interactive-level performance, while others may desire massive scalability or powerful inferencing. Particularly until the point at which we can perform inference in a computationally efficient manner, it is clear that these goals cannot all be satisfied at once in the near future.

My proposal for the near future is the development of a triple store implementing developments of the ideas seen in paper [7], combined with a full backward-chaining OWL reasoner. Forward chaining reasoners cannot scale to massive datasets, so all except for the simplest inferences must be backward chained. Obviously, backward chaining reasoners do not offer anything close to interactive-level performance, but they do not interfere with the store's ability to scale. Key in this idea is the ability to specify whether inference is required or not on the query being performed.

The methods put forward in [7] will require extension to be applicable to a 'live' semantic store, most particularly the ability to update the pattern analysis of data and queries on the fly during operation, and perform reasonably fast transfers to new schema – the refactored information ought to be prepared in the background and brought live during quiet periods

This implementation has several advantages. Scalability is catered for to a significant extent thanks to the use of backward-chained reasoning and application specific schema design. The system maintains the interactive-level performance seen in the best of the current RDF stores when no inference is required on the data, while also offering full inferencing capabilities when they are absolutely required. It should be made clear that this inferencing will not be suitable for interactive-level applications, but it will be particularly suitable for use with agents, since they can simply wait for data to be returned to them.

Resolution for truly massive scalability and distribution will for now have to be dealt with as a different problem. There is clearly a limit to how much data that can be placed on a single server, even if current techniques improve a great deal. There is, then, a clear need for the ability to distribute data across multiple servers while still being able to query across them all. Peer to peer systems as described in [5] offer this capability

Unfortunately, distributed systems are entirely unsuitable for backward chaining reasoning thanks to the latency issues involved with getting triples from stores. It is likely that forward-chained RDFS reasoning will be a practical limit, except where the individual requesting the information is willing to wait a great deal of time.

Another issue with distributing data in a peer to peer fashion is control of data. While peer to peer systems solve the problem of discovering where data is through a globally known hash algorithm, this means that it is impossible to retain ownership of data or restrict access to it, since the data is not necessarily being stored on the server of the individual who owns the data. There is no obvious solution for this problem; as soon as we attempt to store our data only on our server, it becomes difficult for others to

find. This makes it difficult to use a large-scale peer to peer system as a solution for the problem of knowing where data is stored.

Finally, there is the issue of the backing store that the store might use to be considered. This report has evaluated storage systems that use flat files, relational database back ends, XML database back ends, and custom designed file systems indexed using AVL trees. The fastest current designs use relational databases (in particular MySQL 3) to store their data, but the question of whether these databases will continue to provide the requisite scalability as semantic stores develop has to be answered.

While [1] provides a persuasive case for the use of their custom file formats, and [3] provides some evidence for the efficacy of BerkeleyDB, it is clear that they do not yet offer the performance of stores based upon relational databases. What remains to be conclusively proven is whether this is a result of the current immaturity of the mentioned technology versus the massively optimised relational databases, or simply that the use of the relational database is inherently faster. For now, I expect relational databases to remain the backing store of choice for triple stores. In concert with developments of the work in [7], the limitations of relational databases' fixed schema can be overcome while retaining the massive performance bonuses they provide.

6. Evaluation

The Semantic Web as a whole is a truly visionary idea. The potential for increasing knowledge availability and the ability of machines to effectively work with it is enormous. A significant part of realising this potential is the creation of storage systems that can hold and work with the massive amount of data that could be made available. As highlighted in this paper, there are a variety of significant technical challenges to be solved before semantic stores can be considered truly mature.

Unfortunately, the requirements are currently intractable when attempting to solve them all in the context of a single application. Interactive level performance combined with greatly improved scalability is clearly already in reach, and scalability can be further extended (at the cost of some performance) through the use of distribution. Unfortunately, attempts to implement OWL reasoning upon these systems would harm performance and/or scalability.

Until reasoning systems offer massively better performance over large datasets, triple stores with complex reasoners cannot be considered suitable for applications requiring human interaction. The solution presented in this paper offers a compromise that offers the full power of OWL reasoners for applications that do not require interactive-level performance, without preventing those that do from accessing the system without the use of the reasoner.

7. REFERENCES

[1] Adams, T., Gearon, P., Wood, D., Kowari: A Platform for Semantic Web Storage and Analysis. http://www.itee.uq.edu.au/~dwood/docs/www2005-kowari.pdf (May 2005)

[2] Berners-Lee, T., Hendler, J., Lassila, O. The Semantic Web. Scientific American (May 17, 2001)

[3] Bjelogrlic, Z., Gulik, D. W. van, and Reggiori, A. Indexing and Retrieving Semantic Web Resources: the RDFStore model. http://www.w3.org/2001/sw/Europe/events/20031113-storage/positions/asemantics.pdf

[4] Broekstra, J., Harmelen, F van., Kampman, A. Sesame: An Architecture for Storing and Querying RDF Data and Schema Information. Semantics for the WWW. MIT Press, 2001.

[5] Cai, M., Frank, M., RDFPeers: A Scalable Distributed RDF Repository Based on a Structured Peer-to-Peer Network. In P*roceedings of the 13th international conference on the World Wide Web* (New York, USA, May 2004), 650-657.

[6] Carroll, J. J., Dickinson, I., Dollin, C., Reynolds, D., Seaborne, A., Wilkinson, K., Jena: Implementing the Semantic Web Recommendations. In *Proceedings of the 13th international World Wide Web conference on Alternate track papers & posters* (New York, USA, May 2004), 74-83

[7] Ding, L., Kuno, H., Sayers, C., Web, S., Wilkinson, K. Application Specific Schema Design for Storing Large RDF Datasets. In *Proceedings of the First International Workshop on Practical and Scaleable Semantic Systems (PSSS1)* (Sanibel Island, Florida, USA, October 2003)

[8] Gibbins, N., Harris, S. 3store: Efficient bulk RDF storage. In *Proceedings of Knowledge Markup and Semantic Annotation, 3rd International Semantic Web Conference (PSSS '03)* (Hiroshima, Japan), 81-90.

[9] Guo, Y., Heflin, J., Pan, Z., An Evaluation of Knowledge Base Systems for Large OWL Datasets. In *Proceedings of the 3rd International Semantic Web Conference (ISWC2004)* (Hiroshima, Japan, Nov 2004)

[10] Kuno, H., Reynolds, D., Sayers, C., Wilkinson, K., Efficient RDF Storage and Retrieval in Jena2. http://www-scf.usc.edu/~csci586/paper/Wilkinson_etal.pdf

[11] Lee, R., Scalability Report on Triple Store Applications. http://simile.mit.edu/repository/site/reports/stores/stores.pdf