# Hawkeye: A Practical Large Scale Demonstration of Semantic Web Integration

Zhengxiang Pan Abir Qasem Sudhan Kanitkar
Fabiana Prabhakar Jeff Heflin

Department of Computer Science and Engineering, Lehigh University
19 Memorial Dr. West, Bethlehem, PA 18015, U.S.A.
{zhp2, abq2, sgk205, ffp206, heflin}@cse.lehigh.edu

**Abstract.** We discuss our DLDB knowledge base system and evaluate its capability in processing a very large set of real-world Semantic Web data. Using DLDB, we have constructed the Hawkeye knowledge base, in which we have loaded more than 166 million facts from a diverse set of real-world data sources. We use this knowledge base to demonstrate realistic integration queries in e-government and academic scenarios. In order to support Hawkeye, we extended DLDB with additional reasoning capabilities. At present, the Semantic Web consists of numerous independent ontologies. We demonstrate that OWL can be used to integrate these ontologies and thereby integrate the data sources that commit to them. In terms of performance, we show that the load time of our system is linear on the number of triples loaded. Furthermore, we show that many complex queries have response times under one minute, and that simple queries can be answered in seconds.

## 1 Introduction

The 2005 index of Swoogle [2] contained 850,000 of Semantic Web documents, in 2006 this index had 1.5 million SW documents and at the time of writing this paper it boasts a staggering 2.1 million SW Documents. The Semantic Web is growing and clearly scalability is an important requirement for Semantic Web systems. Furthermore, the Semantic Web is an open and decentralized system where different parties can and will, in general, adopt different ontologies. Thus, merely using ontologies, does not reduce heterogeneity: it just raises heterogeneity problems to a different level. Without some form of alignment, the data that is described in terms of one ontology will be inaccessible to users that ask questions in terms of another ontology. In this paper we present a scalable system (166 million triples) and a knowledge base that has been integrated using only OWL axioms (as opposed to special purpose mapping languages).

We put forward that, in addition to providing semantics to the data, OWL can also be used to establish alignments between these heterogeneous web sources. Using map ontologies, ones that contain OWL axioms that align the concepts of two ontologies, we have integrated many autonomous data sources and successfully demonstrated useful queries. For example, consider a researcher looking for colleagues to collaborate with her in a paper. One heuristic she may apply in her search is to look for people who have cited a paper that has been cited by her in her other publications. Obviously, this

can be done using Google, but it will require several intermediate steps to meet her specific information need. Our system can get her the answer from two different sources (Citeseer and DBLP) in just a few seconds. We discuss this query and others that we have tested in Section 3.2

Before we present our work, we would like to briefly review the state of the public Semantic Web. We note that there are several traits that we have observed in the existing Semantic Web data (indexed by Swoogle) that influenced our design choice.

First, we observe that if we account for minor syntactic errors (e.g. missing a type declaration) most of the ontologies in the current Semantic Web have an expressivity equivalent or less than OWL DL. As these syntactic issues can be programmatically resolved, most of the OWL Full ontologies can be easily converted to OWL DL, which is most likely what the developer had intended [1]. In a recent survey of ontologies, Wang et al. [13] report similar syntactic errors leading to OWL Full ontologies. Therefore, our system's overall focus is to support OWL DL as opposed to OWL Full.

Second, we have observed that the ontologies and data from the social network domain are currently dominating the Semantic Web landscape. The most frequently used ontology in the Semantic Web is the Friend of A Friend (FOAF) ontology. It is interesting to note that although FOAF was originally designed for individuals to make their profiles available to public, the prevalence of FOAF data is due to Blog sites and social network sites (LiveJournal, etc.) which generate FOAF data from users' public profile. Each site generates its own URI for an individual and therefore we have several different URIs pointing to the same object. This is essentially an entity resolution problem. In order for us to have a plausible integration of the Semantic Web, we needed to resolve these duplicate entities, establish alignments and add instance equality reasoning to DLDB system. The *owl:InverseFunctionalProperty* has helped us in this task. Basically if a property, p, is annotated as InverseFunctionalProperty, then $\forall$ x, y, z p(y,x) $\wedge$ p(z,x) $\rightarrow$ y = z. With the FOAF data we have used InverseFunctionalProperty to state for example if two individuals (two distinct URIs) have the same email address then they essentially are the same individual.

Third, we have observed that it is important to support the *TransitiveProperty* attribute of OWL properties. There are several ontologies in the Semantic Web that describe properties in terms of this characteristic. For example, many ontologies have made use of transitive properties such as *hasPart* and *subLocationOf*. SKOS, the World Wide Web Consortium's recent effort in describing a controlled vocabulary for thesauri, classification schemes, subject heading systems and taxonomies within the framework of the Semantic Web, makes extensive use of transitive properties

In what follows we first describe our enhanced DLDB system. We present its architecture, design and implementation with a focus on the additional reasoning and optimizations that we have added to the system based upon the characteristics of the Semantic Web. After presenting the system we then describe our Hawkeye knowledge base and at the end present related work and conclude. Note: in this paper we build on our initial work [11] in this area and now present a more comprehensive demonstration on a larger set of Semantic Web data.

## 2 DLDB: A Semantic Web Query Answering System

The initial architecture of DLDB is presented in [10]. It is a knowledge base system that extends a relational database management system with additional capabilities for partial OWL reasoning. The DLDB core consists of a Load API and a Query API implemented in Java. Any DL Implementation Group (DIG) compliant DL reasoner and any SQL compliant RDBMS with a JDBC driver can be plugged into DLDB. This flexible architecture maximizes its customizability and allows reasoners and RDBMSs to run as services or even clustered on multiple machines.

It is known that the complexity of complete OWL DL reasoning is NEXPTime-complete. Our pragmatic approach is to trade some completeness for performance. The overall strategy of DLDB is to find the good balance of precomputation of inference and run-time query execution via standard database operations. Whenever DLDB loads an RDF/OWL file it determines if it is an ontology or instance data document. Ontologies are first processed by the DL reasoner in order to compute implicit subsumptions and the results are used to create tables and views in the database. Instance data are directly loaded into the database tables. The consideration behind this approach is that DL reasoners are optimized for reasoning over ontologies, as opposed to instance data.

DLDB facilitates integration by supporting the perspectives presented in Heflin and Pan [5]. Ontology perspectives allow the same set of data sources to be viewed from different contexts, using different assumptions and background information. Each perspective will be based on an ontology. Thus, data sources that commit to the same ontology have implicitly agreed to share a context. When it makes sense, we also want to maximize integration by including data sources that commit to different ontologies. We require that each Semantic Web query to be associated with a particular perspective, so that the answers to a query depend on the entailment of the perspective. For example, when the perspective is based on different mapping ontologies, a query would get different answers resulting from different mapping axioms.

In DLDB, creating tables corresponds to the definition of classes or properties in ontology. Each class and property has a table named using its URI. This means new tables are created as new ontologies are discovered. In DLDB, class hierarchy information is stored through database views. The view of a class is defined recursively. It is the union of its table and all of its direct subclasses' views. Hence, a class's view contains the instances that are explicitly typed, as well as those that can be inferred.

DLDB currently supports conjunctive queries. In terms of query language, DLDB supports a subset of SPARQL, namely the SELECT query form, combined with the triple pattern and filter that allows numeric types. We think this subset covers most of the frequently posed extensional queries. We plan to support some useful modifiers such as "ORDER BY" and filters on date.

During query execution, predicates and variables in the query are substituted by table names and field names through translation. Depending on the perspective being selected, the table names are further substituted by corresponding database view names. Finally, a standard SQL query sentence is formed and sent to the database via JDBC. Then the RDBMS processes the SQL query and returns appropriate results.

**Instance Equalities.** OWL does not make the unique names assumption, which means that different names do not necessarily imply different objects. Given that many indi-

viduals contribute to the Web, it is highly likely that different IDs will be used to refer to same object.

In DLDB, each unique URI is assigned a unique integer id. Our approach to equality is to designate one id as the canonical id and globally substitute the other id with this canonical id in the knowledge base. The advantage of this approach is that there is effectively only one system identifier for the (known) individual, nevertheless that identifier could be translated into multiple URIs. Since reasoning in DLDB is based on these identifiers instead of URIs, the existing inference and query algorithms do not need to be changed to support equalities.

However, in many cases, the equality information is found much later than the data that it "merges". Thus, each URI is likely to have been already used in multiple assertions. Finding those assertions is especially difficult given the table design of DLDB, where assertions are scattered into a number of tables. It is extremely expensive to scan all the tables in the knowledge base to find all the rows that use a particular id. We devised auxiliary tables to keep track of the tables that each id is appeared in. An `Individual_Occurrence` table is used to record the the occurrences of each id. To substitute an id with another id, the procedure queries those auxiliary tables to find the set of tables (and columns) upon which an update is issued to perform the substitution.

Often times, the knowledge on equality is not given explicitly. Equality could result from inferences across documents: *owl:FunctionalProperty* , *owl:maxCardinality* and *owl:InverseFunctionalProperty* can all be used to infer equalities. DLDB is able to discover equality on individuals using a simple approach. If two URIs have the same value for an *owl:InverseFunctionalProperty*, they are regarded as representing the same individual. A naive approach is to check it every time a value is being inserted into an inverse functional property table. However, this requires a large number of queries and potentially a large number of update operations. In order to improve the throughput of loading, we developed a more sophisticated approach which queries the inverse functional property table periodically during the load. The specific interval is specified by users based upon their application requirements and hardware configurations (we used 1.5 million in our experiment). This approach not only reduces the number of database operations, but also speeds up the executions by bundling a number of database operations as a stored procedure.

**Transitive Closure and Its Interaction with Other Reasoning.** One of the ABox reasoning tasks is to infer implicit property assertions through the transitive property. This task can be regarded as computing a transitive closure over a directed acyclic graph. In DLDB, we must also address how these algorithms interact with other reasoning. One of the existing algorithms is to maintain the transitive closure from scratch, i.e. starting from when the table is empty [3]. Each time a new pair is added, the maintenance algorithm will compute new relations and add them to the table so that the table corresponds to its transitive closure. However, this algorithm would not work under some circumstances. For example, the property *isIn* is transitive, whereas its two subproperties: *isInState* and *isInRegion*, are not (and should not be) transitive. If the data only contains instances of *isInState* and *isInRegion*, no transitive closure algorithm could be invoked. When *isIn* is queried, its transitive closure has not been computed. Note, it is

difficult to continuously maintain the transitive closure of a view because the insertions go through its underlying tables.

Our adapted algorithm, which runs periodically, joins the view iteratively until a fixed point is reached. This algorithm uses a *temp* table to record the results of joining the view of a property with itself and a *delta* table to record the results that are new to the property table. Then the iterations only join the property view with the *delta* table, which is usually much shorter than the property table. This algorithm also takes care of the perspectives, which allows different ontologies to independently describe a property as transitive or not. Details about the algorithms mentioned in this section can be found in our technical report [9].

## 3 The Hawkeye Knowledge Base

This section has two main objectives. First, we want to evaluate the new reasoning capabilities of DLDB. Second, we want to demonstrate the ability to answer realistic queries in the Semantic Web from distributed and heterogeneous data sources. Unfortunately, existing data on the Semantic Web tends to be unrelated. In order to make the queries interesting we needed to augment existing Semantic Web data with some new data sources. We focus on two scenarios which involve data associated with academic publications and government activities.

### 3.1 Data Sources and Maps

Table 1 describes the data sources used in our scenarios. The first column lists the shorthand prefixes we assigned to each data source.

The e-government scenario uses **g** (daml.org), **c** (house.gov), **b** (govtrack.us) and **n** (govtrack.us) data sources. The academic publication scenario uses **d** (dblp.uni-trier.de), **s** (citeseer.ist.psu.edu), **a** (aigp.csres.utexas.edu), **w** (nsf.gov) and **f** (found from Swoogle's crawl) data sources.

To augment our Semantic Web data we transformed the data from these sources to RDF which commits to valid ontologies. For sources originally in XML format, we developed an ontology for each of them based on their XML schema and developed domain specific scripts to translate the XML to conforming RDF. For each of those sources originally in HTML pages, we developed a crawler to collect the pages and a scraper which extracts the desired information from these pages. We then developed ontologies for each of them and generated conforming RDF for the scraped data.

The purpose of using data from multiple sources is that a single source doesn't hold all the information for a certain individual. When different URIs are used to refer the same individual, we must explicitly annotate that they are equivalent using the owl:sameAs property. There are a number of techniques with varying accuracy for automatic co-reference resolution. We try to match the names of individuals (e.g. Authors) from each of the sources using simple string matching techniques. In the case of FOAF, our system relies on inverse functional properties to infer equivalence of individuals. Totally, we created 109,790 sameAs statements between 4 pairs of data sources.

| Pre | Data Source | Original Format | Classes | Properties | Triples |
|---|---|---|---|---|---|
| a | AIGP | No Ontology<br>Set of HTML Pages | AIResearcher | hasAdvisor<br>influencedBy<br>hasInfluenced | 5973 |
| b | Bill Data | No Ontology<br>RDF | Politician<br>Bill | name<br>sponsoredBy | 75711 |
| c | 107th Congress | No Ontology<br>XML Data | Member<br>USCD | party, isIn<br>fromUSCD | 2628 |
| d | DBLP | No Ontology<br>XML Data | Article<br>foaf:Person | author<br>coauthor | 15523209 |
| f | FOAF | RDF Schema Ontology | Person | knows | 11601453 |
| g | Geographic Data | DAML Ontology<br>DAML Data | USRegion<br>USState | memberstate<br>region | 578 |
| n | Census Data | No Ontology<br>N3 File | State | population<br>landarea | 314 |
| s | Citeseer | No Ontology<br>XML Data | Article<br>foaf:Person | author, coauthor<br>references | 7630021 |
| w | NSF Awards Data | No Ontology<br>Set of HTML Pages | NSFAward | principalInvestigator<br>state | 462102 |

**Table 1.** Data sources summary

To use the concepts and properties of different ontologies, we must explicitly specify the relationships between them. For this we need to use OWL axioms in a separate ontology which we call the map ontology. For example, for the academic domain the property *dc:creator* used in the Citeseer ontology is a super property of the property *author* used in the DBLP ontology.

### 3.2 Performance

We have used Swoogle's 2006 index as our dataset along with the data we prepared that described above. The DLDB main program runs on a workstation featuring dual 64-bit CPUs and 10GB main memory. The RDBMS is MySQL 5.0 and the DL reasoner is RacerPro. It took 650 hours to process the 1.7 million urls from Swoogle. Many of these had not been successfully downloaded due to various network issues, such as HTTP404 and connection timed out. We successfully retrieved 759,834 SW documents; the time to load and process just these documents is about 350 hours. In total 16,280 among them are identified as ontologies by Hawkeye. It takes approximately 18 gigabytes disk space for the RDBMS to store the 166M resulting triples. To the best of our knowledge this is the largest load of diverse, real-world Semantic Web data. This once again validates that the DLDB approach scales fairly well.

**Load Performance.** The chart in Figure 1 shows the cumulative load time after each million triples loaded into the system. In general we see that the load time increases gradually and our system scales well. Note the "local" time is defined as the total time minus the time spent in transferring the documents from a remote host. The "limited

reasoning" time is the local time minus the time spent in batch processing of ABox reasoning (including *InverseFunctionalProperty* and transitive closure inference). The steep slope from 43 to 49 million triples corresponds to the identification of a bug in our code and its subsequent correction. The steep slope at the end of the curves is contributed by a large number of explicit sameAs statements in the DBLP and Citeseer data. These statements were loaded after the data they mapped, thereby requiring a lot of substitutions. In a typical data load, the sameAs statements would be interspersed with the data and processing would be faster.

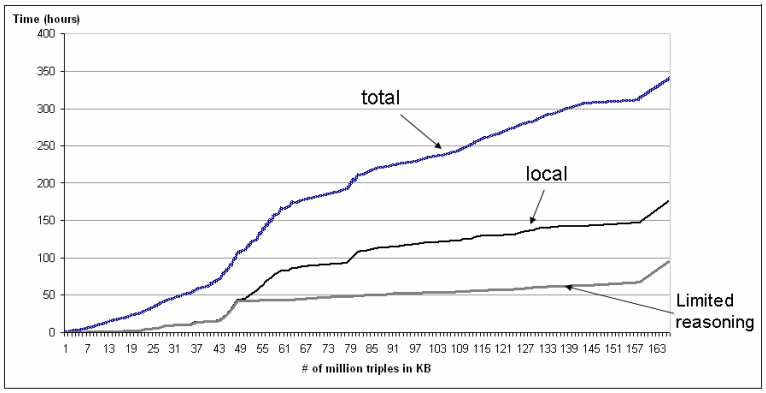


**Fig. 1.** Hawkeye Cumulative Load time

**Query Performance.** In order to evaluate the query performance of Hawkeye, we used the six query templates described in Table 2. For each query template, we listed the inferences and the data sources. Note the mappings between different sources are indispensable too. Table 3 shows the query performance of our system. For each query template, we issued a number of variations by changing the constants in the query. We then calculated the average and standard deviation of the response time. Most of the queries finished very quickly (in less than a minute). We also calculated the percentage of the non-zero answers and the average number of results for each query template. The high percentages of the non-zero answers demonstrates a significant degree of integration of the sources, particularly with respect to entity resolution. If our mapping ontologies or individual maps were insufficient, we would get many (perhaps even all) queries with 0 answers.

## 4 Lessons Learned

In working with real Semantic Web data, we have encountered many interesting challenges. In this section, we summarize those challenges and our experiences.

In looking at the real data and our experiment, we found out that for a practical Semantic Web knowledge base system, the key inference capabilities are individual

| Query | Description | Inferences | Sources |
|---|---|---|---|
| Pol1 | Find a politician from a region who has sponsored a bill in some specific topic | subClassOf TransitiveProperty | g,c,b |
| Pol2 | Find bills sponsored by a given politician and the population of his state | equivalentClassOf | c,b,n |
| Pol3 | Name of the politicians who come from a certain region | TransitiveProperty | g,c |
| Aca1 | Find articles written by a Professor's advisees | sameAs | d,s,a |
| Aca2 | Find people who I know who have cited a paper also cited by me | sameAs, inverseOf InverseFunctionalProperty | d,s,f |
| Aca3 | Find academic influence of a researcher (articles written by all people in the advisee chain) | sameAs TransitiveProperty | d,s,a |
| Aca4 | Find publications of the AIResearchers from a certain state who have been awarded NSF grants | sameAs subClassOf | d,s,a,w |

**Table 2.** Query descriptions

| | | Response time (ms) | | No. of results | |
|---|---|---|---|---|---|
| Query template | No. of variations | Avg. | Stdev. | %of non-zeros | Avg |
| Pol1 | 300 | 871 | 635 | 90 | 63 |
| Pol2 | 300 | 1018 | 897 | 100 | 115 |
| Pol3 | 300 | 53 | 22 | 95 | 89 |
| Aca1 | 200 | 24151 | 632 | 95 | 326 |
| Aca2 | 20 | 345906 | 32149 | 87 | 1626 |
| Aca3 | 200 | 24659 | 654 | 95 | 3549 |
| Aca4 | 51 | 25318 | 2063 | 90 | 10245 |

**Table 3.** Query performance

equivalence (both explicit and inferred), subclass / subproperty inference, inverse property, transitive property. These were sufficient to do integration and testing of the many web sources that we looked at. Note, arithmetic would probably be critical too (for unit conversion), but OWL doesn't support it.

Our DLDB design occasionally forms queries that the underlying database management system finds difficult to optimize. For example, one of our queries (Aca 2 in Table 2) took about 6 minutes to complete. This is due to the fact that it uses the foaf:knows property which has 16 million instances in our database. We performed some tests where we manually re-wrote the view for the foaf:knows property. We noticed that the query over this view had a constant expression in the where clause that would reduce the view scope. By moving the constant expression from the where clause of the query to the where clause of the view we were able to reduce the query execution to 3 minutes. We believe that future work can be done in order to create an algorithm to automatically optimize the queries which will increase the performance considerably.

During the implementation of DLDB, we observed that if precomputation of ABox reasoning is needed, it is faster to do it in batch, rather than doing it per assertion. For

example, when discovering individual equalities, our batch approach not only reduces the number of database operations, but also speeds up the executions by bundling a number of database operations as a stored procedure.

## 5 Related Work

We claim that this is the first attempt to load the real Semantic Web data into a single knowledge base system. We should however note that there are several projects that process the Semantic Web in various other ways. For example, Swoogle [2] is the largest index of Semantic Web documents. However, Swoogle's query and retrieval mechanism is basically an information retrieval system. This does not exploit the reasoning that can be done over Semantic Web data.

There are some ongoing efforts to bootstrap the Semantic Web by providing ontologies and reusable knowledge bases, such as TAP [4]and "CS AKTive Space" [12]. Although they have large amounts of data, they each assume a common ontology.

In the past few years there has been a growing interest in the development of systems that will store and process large amount of Semantic Web data. The general design approach of these systems is similar to ours, in the sense that they all use some database systems to gain scalability while supporting as much inference as possible by processing and storing entailments. However, most of these systems emphasize RDF and RDF(S) data at the expense of OWL reasoning. Some systems resemble the capabilities of DLDB, such as KAON2 [6], which uses a novel algorithm to reduce OWL DL into disjunctive datalog programs. OWLIM [7] uses a rule engine to support a limited OWL-Lite reasoning. Minerva [14] uses DL reasoner to do TBox reasoning and a rule engine to do ABox reasoning. It is claimed to be sound and complete on DHL (a subset of OWL-DL) ontologies and reportedly less scalable than DLDB in terms of load time [14]. Both OWLIM and Minerva chose the "vertical" table design. To the best of our knowledge, none of the systems above have been used with a real world Semantic Web data at this scale (166M triples) , though BigOWLIM [8] has been claimed to support 1 billion triples of artificially generated data.

## 6 Conclusion and Future Work

In this paper we present an enhanced version of our DLDB system.We have extended our previous work by identifying and implementing critical inference capabilities and optimizing the system so that it can now handle at least 166 million facts as opposed to the 45 million of the previous version. The performance on query response time remains highly scalable, most of the queries in our experiment can be finished in less than one minute. We use ontology alignments expressed in OWL to provide a uniform view of the Semantic Web to the user We defer integration until query time and thus provide a framework where the user is not bound by a predetermined schema. Our ontology perspective mechanism gives the user the flexibility to choose the type of integration (s)he desires. Our maps do not need any additional language primitives beyond OWL. Therefore the maps as created and published become part of the Semantic Web. We put forward that scalability is more critical in processing the data sources as opposed

to ontologies, because data sources will substantially outnumber the ontologies in the Semantic Web.

Although we believe our work is a first step in the right direction, we have discovered many issues that remain unsolved. First, although our system scales well to the current size of the Semantic Web, it is still unknown if such techniques will continue to scale well as the Semantic Web grows. Second, we will investigate query optimization techniques that can improve the query response time.

## 7 Acknowledgment

This material is based upon work supported by the National Science Foundation (NSF) under Grant No. IIS-0346963. We sincerely thank Tim Finnin of UMBC for providing us access to Swoogle's index of URLs.

## References

1. S. Bechhofer and R. Volz. Patching syntax in OWL ontologies. In *Proceedings of the Third International Semantic Web Conference*, 2004.
2. L. Ding, T. Finin, A. Joshi, Y. Peng, R. Pan, and P. Reddivari. Search on the semantic web. *IEEE Computer*, 10(38):62–69, October 2005.
3. G. Dong, L. Libkin, J. Su, and L. Wong. Maintaining transitive closure of graphs in SQL. *Int. Journal of Information Technology*, 1999.
4. R. Guha. Tap: Towards the semantic web. Demo on World Wide Web 2002 Conference, 2002. At: http://tap.stanford.edu/www2002.ppt.
5. J. Heflin and Z. Pan. A model theoretic semantics for ontology versioning. In *Proc. of the 3rd International Semantic Web Conference*, pages 62–76, 2004.
6. U. Hustadt, B. Motik, and U. Sattler. Reducing shiq description logic to disjunctive datalog programs. In *Proc. of the 9th International Conference on Knowledge Representation and Reasoning*, pages 152–162, 2004.
7. A. Kiryakov. Owlim: balancing between scalable repository and light-weight reasoner. In *Developer's Track of WWW2006*, 2006.
8. D. Ognyanoff, A. Kiryakov, R. Velkov, and M. Yankova. A scalable repository for massive semantic annotation. Technical Report D2.6.3, SEKT project, 2007.
9. Z. Pan et al. Hawkeye: A practical large scale demonstration of semantic web integration. Technical Report LU-CSE-07-006, Lehigh University, 2007.
10. Z. Pan and J. Heflin. DLDB: Extending relational databases to support semantic web queries. In *Proc. of the Workshop on Practical and Scaleable Semantic Web Systms, ISWC*, pages 109–113, 2003.
11. Z. Pan, A. Qasem, and J. Heflin. An investigation into the feasibility of the semantic web. In *Proc. of Twenty First National Conference on Artificial Intelligence (AAAI 2006)*, 2006.
12. M. C. Schraefel, N. R. Shadbolt, N. Gibbins, S. Harris, and H. Glaser. CS AKTive space: representing computer science in the semantic web. In *WWW '04: Proceedings of the 13th international conference on World Wide Web*, 2004.
13. T. D. Wang, B. Parsia, and J. Hendler. A survey of the web ontology landscape. In *Proc. of the 5th Int. Semantic Web Conference (ISWC 2006), Athens, Georgia*, 2006.
14. J. Zhou, L. Ma, Q. Liu, L. Zhang, Y. Yu, , and Y. Pan. Minerva: A scalable owl ontology storage and inference system. In *Proc. of Asia Semantic Web Conference (ASWC)*, pages 429–443, 2006.