

Hawkeye: A Practical Large Scale Demonstration of Semantic Web Integration

Lehigh University Technical Report LU-CSE-07-006

June 8th, 2007

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. At present, the Semantic Web consists of numerous independent ontologies. We put forward that OWL can be used to integrate these ontologies and thereby integrate the data sources that commit to them. In this paper we present the Hawkeye knowledge base, in which we have loaded more than 166 million facts from a diverse set of real-world data sources. In order to support Hawkeye, we extended our DLDB knowledge base system with additional reasoning capabilities. DLDB is a system that given sufficient OWL descriptions, can answer queries that span heterogeneous data sources. We use the Hawkeye knowledge base to demonstrate realistic integration queries in e-government and academic scenarios. For example, our system can produce answers that integrates Citeseer and DBLP. We achieve this integration in a declarative way by only using OWL. These queries cannot be answered by traditional search engines. 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 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. We argue that, in addition to providing semantics to the data, OWL can also be used to establish alignments between these heterogeneous web sources. This alignment is essentially a set of logical statements that establish correspondence between ontologies. In this paper we present a system that uses ontology alignments expressed in OWL to provide a uniform view of the Semantic Web to the user. We build on our initial work [24] in this area and now present a more comprehensive demonstration on a larger set of Semantic Web data. Specifically we make the following new contributions:

1. Based on real world scenarios we identify critical inference capabilities needed and implement them in our system.

2. We experiment with a much larger Semantic Web dataset and implement some optimizations to improve performance with this larger data set.
3. We perform a detailed and extensive demonstration that includes integration of traditional web sources into the system.

Although we discuss the uniqueness of our approach in the related work section, we feel it may be useful to mention some aspects of our work early on. First, we note that establishing alignment between ontologies is a non-trivial problem. However, our work is not about ontology alignment. There is a well-established body of research in the area of automated ontology alignment [7]; although our work will benefit from the results of this research, it is not predicated upon its success. Instead, we see the potential of a network effect that arises when many individuals each contribute a few small maps. As we use OWL to articulate alignments, and since OWL is shareable via web, any one can create alignments and make it available for others to use. By embodying a web like framework in the alignment (mapping), we hypothesize that integration will be an emergent property of the Semantic Web. Note: obviously we will not have alignments between all pairs of ontologies. However it should be possible to compose an alignment from existing alignments by traversing through a “semantic connection” of alignments.

We use a centralized repository approach to storing the Semantic Web data. In traditional information integration research, the query is rewritten into queries that are distributed to the sources, thereby assuring the most current answers. However, in this work we decided to accept a limited staleness of the data in favor of efficiency. The success of contemporary search engines in providing fairly fresh data, suggests that crawling and storing data in a centralized repository is a viable approach, as long as the sources are not highly dynamic. We should also note that unlike the data warehousing approach [33] we do not import our data to a pre-designed schema. As per the principle of dataspace systems[14], the actual integration is still done at query time. We merely load all the data and the alignments (which is just more OWL) at system startup. As we do the integration at query time, the user is not bound by a predetermined alignment and can select or reject an alignment based on her world view. This flexibility is not only desirable but essential in the Web environment.

In what follows we first describe the nature of the data sources in the current Semantic Web. Some of these characteristics have significantly influenced the design and optimization of our system. We then describe our integration framework. We subsequently present 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.

2 The Semantic Web Landscape

In this section we briefly review the state of the public Semantic Web. Swoogle [9] is the largest index of Semantic Web documents. The 2006 index of Swoogle data contained 22,468 ontologies and 1.5 million Semantic Web documents. The Semantic Web is still in its early stage of evolution and the nature of the current Semantic Web data

reflects this nascent state. For example, much of the data only commits to a handful of ontologies. In 2006, 25% of the data files found by Swoogle each committed to one of five different ontologies. A large number of ontologies do not have any data that commits to them. This is probably because these ontologies have been created for exposition purposes and have never seen any real use. The nature of the Semantic Web data prompted us to make some functionality enhancements in the DLDB system. We will describe the enhancements in Section 3 but here we briefly describe the particular characteristics that prompted those enhancements and provide an overview of what we have done to address them.

First, we observe that if we account for minor syntactic errors (e.g. missing a class 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 [2]. In a recent survey of ontologies, Wang et al. [31] 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, there is a dearth of individually created FOAF documents in the Semantic Web. The prevalence of FOAF data is due to Blog sites and social network sites (LiveJournal, etc.) which generate FOAF data from the user's 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 [32] (also known as co-reference problem in natural language processing). 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 InverseFunctionalProperty of OWL 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 implement the TransitiveProperty attribute of OWL properties. There are several ontologies in the Semantic Web that describe properties in terms of this characteristic. For example we have observed that properties like hasPart (describing a composition) and subLocationOf (describing photos and its regions) have made use of transitive properties. 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 [30].

In addition, we have identified several data sources where addition of this property allows us to issue powerful queries. Consider for example the Artificial Intelligence Genealogy Project (AIGP) website. This site has a page for each academic Artificial Intelligence researcher where it lists her adviser(s) and advisee(s). Now consider we

want to find out the academic lineage of Marc Goodman. In order to trace this lineage we have to first go to his adviser’s page and then from their his adviser’s adviser’s page etc. until we reach the top most node, which in this case is Marvin Minsky. In OWL we can describe a super property `academicAncestor` of this adviser relationship, and state that it is a `TransitiveProperty`. Now we can ask a query that will give us all the researchers by whom Marc Goodman is influenced, i.e. his academic lineage.

3 Perspectives: An Integration Framework

In prior work, we have defined ontology perspectives which allows the same set of data sources to be viewed from different contexts, using different assumptions and background information [15]. That work also presents a model theoretic description of perspectives. In this section, we set aside the versioning issues of that paper and introduce some essential definitions.

Each perspective will be based on an ontology, hereafter called the basis ontology or base of the perspective. By providing a set of terms and a standard set of axioms, an ontology provides a shared context. 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 now provide informal definitions to describe our model of the Semantic Web. A Semantic Web space \mathcal{W} is a pair $\langle \mathcal{O}, \mathcal{S} \rangle$, where \mathcal{O} is a set of ontologies and \mathcal{S} is a set of data sources.

Definition 1 *An ontology O in \mathcal{O} is a four-tuple $\langle \mathcal{C}, \mathcal{R}, \mathcal{T}, \mathcal{E} \rangle$, where*

1. \mathcal{C} is a set of concepts
2. \mathcal{R} is a set of roles
3. \mathcal{T} is a TBox that consists of a set of axioms
4. $\mathcal{E} \subset \mathcal{O}$ is the set of ontologies that are extended by O .

An ontology defines a set of concepts and a set of roles, the union of which is referred as vocabulary. It also contains a set of axioms, which is called TBox. An ontology can extend another, which means that it adds new vocabulary and or axioms. Extension is sometimes referred to as inclusion or importing.

For convenience, we define the concept of an ancestor ontology. An ancestor of an ontology is an ontology extended either directly or indirectly by it.¹ If O_2 is an ancestor of O_1 , we write $O_2 \in \text{anc}(O_1)$. The formal definition of an ancestor is:

Definition 2 (Ancestor function) *Given a semantic web space $\mathcal{W} = \langle \mathcal{O}, \mathcal{S} \rangle$ and two ontologies $O_1 = \langle \mathcal{C}_1, \mathcal{R}_1, \mathcal{T}_1, \mathcal{E}_1 \rangle$ and $O_2 = \langle \mathcal{C}_2, \mathcal{R}_2, \mathcal{T}_2, \mathcal{E}_2 \rangle$, $O_2 \in \text{anc}(O_1)$ iff*

1. $O_2 \neq O_1$
2. $O_2 \in \mathcal{E}_1$ or $\exists O_i$ such that $O_i \in \mathcal{E}_1 \wedge O_2 \in \text{anc}(O_i)$.

¹ Extension is sometimes referred to as inclusion or importing. The semantics of our usage are clarified in Definition 3

Note the ancestor function returns the extension closure of an ontology, which does not include the ontology itself.

For ontology extension to have its intuitive meaning, all models of an ontology should also be models of every ontology extended by it.

Definition 3 Given an ontology $O = \langle \mathcal{C}, \mathcal{R}, \mathcal{T}, \mathcal{E} \rangle$, an interpretation \mathcal{I} is a model of O iff \mathcal{I} satisfies every axiom in \mathcal{T} and \mathcal{I} is a model of every ontology O_i such that $O_i \in \mathcal{E}$.

We now define the semantics of a data source.

Definition 4 A data source s in \mathcal{S} is a pair $\langle \mathcal{A}, O \rangle$, where

1. \mathcal{A} is a ABox that consists of a set of formulas
2. $O \in \mathcal{O}$ is the ontology that s commits to.

When a data source commits to an ontology, it has agreed to the terminology and definitions of the ontology. It means that for data source $s = \langle \mathcal{A}_s, O_s \rangle$, all the concepts and roles that are referenced in \mathcal{A}_s should be either from the ontology O_s or $anc(O_s)$. Thus every interpretation that is a model of data source must also be a model of the ontology for that data source.

Definition 5 Let $s = \langle \mathcal{A}, O \rangle$ be a data source commits to ontology O and \mathcal{I} be an interpretation. \mathcal{I} is a model of s iff \mathcal{I} is a model of O and \mathcal{I} satisfies every formula in \mathcal{A} .

We now define an ontology perspective model of a Semantic Web space:

Definition 6 (Ontology Perspective Model) An interpretation \mathcal{I} is an ontology perspective model of a semantic web space $\mathcal{W} = \langle \mathcal{O}, \mathcal{S} \rangle$ based on $O \in \mathcal{O}$ (written $\mathcal{I} \models_O \mathcal{W}$) iff:

1. \mathcal{I} is a model of O
2. for each $s = \langle \mathcal{A}_s, O_s \rangle \in \mathcal{S}$ such $O_s = O$ or $O_s = anc(O)$, \mathcal{I} is a model of s .

We now turn our attention to how one can reason with ontology perspectives.

Definition 7 (Ontology Perspective Entailment) Let $\mathcal{W} = \langle \mathcal{O}, \mathcal{S} \rangle$ be a semantic web space and O be an ontology. A formula ϕ is a logical consequence of the ontology perspective of \mathcal{W} based on O (written $\mathcal{W} \models_O \phi$) iff for every interpretation \mathcal{I} such that $\mathcal{I} \models_O \mathcal{W}$, $\mathcal{I} \models \phi$.

Theoretically, each of these perspectives represents a set of beliefs about the state of the world, and could be considered a knowledge base. Thus, the answer to a semantic web query must be relative to a specific perspective. We now define a Semantic Web query that is similar to the definition of conjunctive queries for DL as described by Horrocks and Tessaris[16].

Definition 8 Given a Semantic Web Space $\langle \mathcal{O}, \mathcal{S} \rangle$, a Semantic Web query is a pair $\langle O, \rho \rangle$ where $O \in \mathcal{O}$ is the base ontology of the perspective and ρ is a conjunction of query terms q_1, \dots, q_n . Each query term q_i is of the form $x:c$ or $\langle x, y \rangle:r$, where c is a concept and r is a role from O or ancestor of O and x, y are either individual names or existentially quantified variables.

An answer to the query $\langle O, \rho \rangle$ is θ iff $\mathcal{W} \models_O \theta \rho$ where θ is a substitution for the variables in ρ .

In order to relate our framework to ontology perspectives theory, we now define the mapping ontology between a sequence of ontologies O_i, \dots, O_n as O_m , where O_m extends O_i, \dots, O_n and contains the axioms that map their vocabularies.

Let's take an example to see how our definitions can be applied to the real ontologies. Imagine we have two simple ontologies. O_1 defines concept `Car` and O_2 defines concept `Automobile`. Two data sources commits to them respectively: r_1 states $\{ezz3290 \in O_1 : Car\}$ and r_2 states $\{dfg2134 \in O_2 : Automobile\}$. A map between O_1 and O_2 would be an ontology O_{m12} that imports O_1, O_2 and has the axiom $\{O_1 : Car \Leftrightarrow O_2 : Automobile\}$. We have a query formula $Car(x)$. If the query's perspective is based on either O_1 or O_2 , there would be one or zero answers. However, by choosing O_{m12} as the perspective, the query will retrieve both instances.

We argue that our perspectives have at least two advantages over traditional knowledge representation languages. First, the occurrence of inconsistency is reduced compared to using a global view, since only a relevant subset of the Semantic Web is involved in processing a query. Even if two ontologies have conflicting axioms, inconsistency would not necessarily happen in most perspectives other than the ones that are based on the common descendants of the conflicting ontologies. Second, the integration of information resources is flexible, i.e. two data sources can be included in the same perspective as long as the ontologies they commit to are both being extended by a third ontology.

4 DLDB: A Semantic Web Query Answering System

4.1 Initial Architecture

The initial architecture of DLDB is presented in [23]. It is a knowledge base system that extends a relational data-base management system with additional capabilities for partial OWL reasoning. In this section, we briefly review the aspects of DLDB that were presented in our earlier work [24]. As shown in Figure 1, the components of DLDB are loosely coupled. 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 [1]. Our pragmatic approach is to trade some completeness for performance. The overall strategy of DLDB is to find the ideal 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.

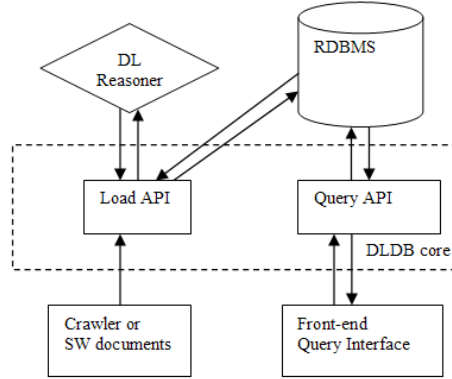


Fig. 1. The Architecture of DLDB

Performance Analysis. Our early experiments suggested that DLDB was a good candidate for a large scale Semantic Web repository [12]. We tested DLDB using LUBM, a benchmark suite we designed for Semantic Web knowledge base systems. LUBM has also been used significantly by others in the community. In this experiment, we have tested systems of different reasoning capabilities.

Figure 2 shows the load times. It turned out that the scalability of DLDB is very good when compared to Sesame [5] and OWLJessKB [19].

Figure 3 shows that DLDB actually achieves good balance between query response time and completeness. The fourteen queries in the benchmark were chosen to cover a range of properties in terms of input size, selectivity, complexity, assumed hierarchy and assumed logical inference. Note the percentage of the filled area of each column is to indicate the degree of completeness of the corresponding system. The degree of completeness is defined as the percentage of entailed answers that are returned by the system.

Based on the benchmark results presented in [12] and the availability of a large amount of real Semantic web data in recent years, we felt that the time was right to test the DLDB model in a large real-world setting.

Table Design. For the basic table design of DLDB, we adopted an approach that is similar to the “Schema-aware” representation in [29]. According to this approach,

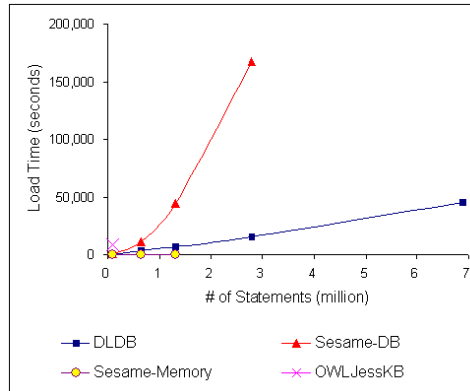


Fig. 2. Load time of DLDB and Other Systems (from [12])

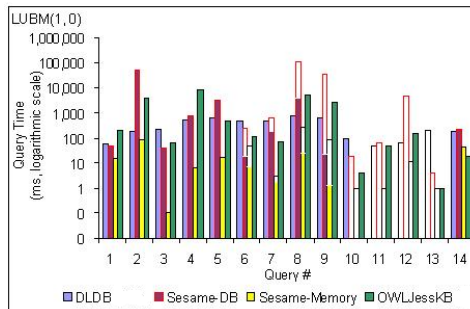


Fig. 3. Query Response Time and Completeness (from [13])

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.

Since each row in the class tables corresponds to an instance of the class, an 'ID' field is needed here to record the ID of the instances. The rest of the data about an instance is recorded using the table per property (a.k.a. decompositional) approach. Each instance of a property must have a subject and an object, which together identify the instance itself. Thus the 'subject' and 'object' fields are set in each table for property. Normally, the 'subject' and 'object' fields are foreign keys from the 'ID' field of the class tables that are the domain or range of the property, respectively. However, if the property's range is a declared data type, then the 'object' field is of the corresponding data type (RDF and OWL use XML Schema data types). Currently Hawkeye supports integer and float in addition to string.

When DLDB receives a data document, the data is stored only in the tables that correspond to the relevant classes and properties. In comparison, the vertical (also called "schema-oblivious") approach [29] uses a single table for storing both RDF/S schemata and resource descriptions under the form of triples (subject-predicate-object).

The table design of DLDB has two advantages over the vertical approach. First, it will contain smaller tables than the vertical scheme. Second, it preserves the data types of the properties and hence can directly and efficiently answer queries involving numeric comparison, such as finding individuals whose ages are under 21. Compared to the traditional relational model, where properties correspond to columns, multi-valued properties (attributes) in DLDB don't need to be identified at ontology design time.

In addition to the basic table design, some details should be taken into account when implementing the database schemas for the system. First, we need a scheme for naming the class and property tables. Note, using the full URI will not work, because these URIs often exceed the RDBMS's limits on the length of table names. However, the local name is also insufficient because many ontologies may include the same local name. So we assign a unique sequence number to each loaded ontology. Then each table's name is a class or property's local name plus its ontology's sequence number. This is supported by an extra table:

```
ONTOLOGIES_INDEX(Url, SeqNum)
```

that is used to register and manage all the ontologies in the knowledge base. The sequence number will be assigned by the system when an ontology is first loaded into the database.

Sometimes it is important to know which document a particular statement came from, or how many documents contain a particular statement. We support this capability by including a 'source' field in each class and property table. Together with other fields, it serves as the multiple-field primary key of the table. In other words, the combination of all the information of one record identifies each instance stored in the knowledge base. An example of class and property tables might be:

```
STUDENT:1(ID, Source)
```

```
TAKESCOURSE:1(Subject, Object, Source)
```

In order to shrink the size of the database and hence reduce the average query time, Hawkeye assigns each URI a unique numeric ID in the system. We use a table:

URI_INDEX (Uri, Id)

to record the URI-ID pairs. Thus, for a particular resource, its URI is only stored once; its corresponding ID number will be supplied to any other tables. Since the typical URI is often 20-50 characters long and the size of an integer is only 4 bytes in the database, this leads to a significant savings in disk space utilized. Furthermore, query performance is also improved due to the faster joins on integers instead of strings and the reduced table size. By discriminating the DataType properties and ObjectType properties, the literals are kept in their original form without being substituted by ID numbers. The reason why we don't assign IDs to literals is 1) literals are less likely to be repeated in the data; and 2) literals are less likely to be joined with other tables because they are never used as keys.

Unsurprisingly, the tradeoff of doing the URI-ID translation is an increase in load time. We use a hash table to cache URI-ID pairs found recently during the current loading process. Since URIs are likely to repeat in one document or neighboring documents, this cache saves a lot of time by avoiding lookup queries when possible.

Supporting OWL entailment and perspectives. 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.

Using OWL's class constructors and property restrictions, a DL reasoner can compute class subsumption, i.e., the implicit *rdfs:subClassOf* relations. When presented with an ontology, Hawkeye uses a DL reasoner to precompute subsumption, and then uses the inferred class taxonomy to create class views. Thus, Hawkeye only consults the DL reasoner once for each new ontology in the knowledge base. Whenever queries are issued concerning the instances of the ontology, the inferred hierarchy information can be automatically utilized. This form of precomputation allows us to make use of important inferences without increasing a penalty at query time. However, as we will see later this approach can miss significant deductions.

Following the definitions of perspectives, every time DLDB loads a new ontology *O*, it actually sends *O* with all the ancestors of *O* to the reasoner for precomputation. Since different perspectives may have different axioms associated with a class / property, the subsumption hierarchy of a class/property varies across different perspectives. In addition, data may or may not be visible to a particular perspective depending on the ontology it commits to. These requirements pose significant challenges to the design of DLDB. Our solution is to utilize database views to represent perspectives. DLDB creates views on each ancestor ontology's classes and properties but does not create additional tables for them. When a document contains data for a class or property defined in another ontology, that data is stored in the table of the class or property and a special column is used to indicate which ontology the data committed to. The view of a class or property defined in an ancestor ontology reflects the axioms in the ancestor ontology as well as the current ontology that imports the ancestor ontology. The views are virtual queries, so that the number of views has little impact on the size of the database. Meanwhile, the query based on different perspectives will be translated into corresponding database views.

However, without the separate tables, the instances that commit to different ontologies are indistinguishable. This is problematic as the perspective model does not include the data sources that commit to a descendant ontology. To solve this problem, we add a new field in every class table, namely the 'onto' field. Thus, each instance of a particular class is recorded by its ID, its source document's ID and the ID of the ontology to which the data source commits. The example tables would be

```
O1:Student (ID, Source, Onto)
```

```
O1:takesCourse (Subject, Object, Source, Onto)
```

Consider that in ontology *O1* the class *Student* is defined as all the people who take a Course, the class *GradStudent* is defined as all the people who take a *GradCourse*, and the class *GradCourse* is a subclass of *Course*. Then, by OWL entailment we can conclude that *GradStudent* is a subclass of *Student*. Thus, if we call the class view creation algorithm after interaction with a DL reasoner, the view of *Student* will be defined as:

```
CREATE O1:Student:viewBy:O1 AS
SELECT * FROM O1:Student WHERE O1:Student.Onto=O1
UNION SELECT * FROM
O1:UndergraduateStudent:viewBy:O1
UNION SELECT * FROM
O1:GradStudent:viewBy:O1;
```

Note the first token in the name of a view represents the ontology that defines the term and the last token in the name represents the ontology that causes the view to be created. So another ontology *O2* that imports *O1* will include a view *O1:Student:viewBy:O2*. This view will differ from the *O1:Student:viewBy:O1* in that it will include data that commits to *O2*.

Since the views are defined recursively, the indirect subsumptions can be automatically taken care of by the database system. For example, the view of *GradStudent* can be defined as:

```
CREATE O1:GradStudent:viewBy:O1 AS
SELECT * FROM O1:GradStudent
UNION SELECT * FROM
O1:PhDStudent:viewBy:O1;
```

The queries on *Student* will return members of *PhDStudent* although *PhDStudent* is not in the view definition of *Student*.

The latest DLDB distribution is implemented as a storage model in the HAWK[22] framework. HAWK is a repository framework and toolkit that supports OWL. It provides APIs as well as implementations for parsing, editing, manipulating and preserving of OWL ontologies.

Query Answering. The query API of DLDB receives queries from users, executes query operations and returns query results to them. This API currently supports conjunctive queries. During 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.

Since we build the class and property hierarchy when loading the ontology, there is no need to call the DL reasoner at query time. The results returned by the RDBMS can be directly served as the answer to the original query. We think this approach will make the query answering system much more efficient than conducting DL reasoning at query time.

Let's take a sample query to illustrate how the process works. The user's original query is "find all graduate students who take course0 at foo University, from the perspective of O1." The query expressed in SPARQL [25]:

```
PREFIX univ: <O1>
SELECT ?x
WHERE {rdf:type univ:GraduateStudent ?X.
univ:takesCourse ?X http://www.foo.edu/course0}
where question marks indicate variables.
```

First the predicates and variables are translated into table names, field (column) names and conditions through the query translation algorithm. Then the URI-index table is taken into account to retrieval the URIs instead of internal IDs. Finally, that example query is translated into SQL sentences like:

```
SELECT DISTINCT ui0.url
FROM O1:GradStudent:viewBy:O1, uri_index AS ui0,
O1:takescourse:viewBy:O1, uri_index AS ui01
WHERE O1:GradStudent:viewBy:O1.id = ui0.id
AND ui01.url='http://www.foo.edu/Course0'
AND O1:takescourse:viewBy:O1.object=ui01.id
AND O1:GradStudent:viewBy:O1.id
=O1:takescourse:viewBy:O1.subject;
```

4.2 Enhancements to Reasoning

In Section 2, we described some forms of reasoning that had not been supported by the initial design of DLDB. In this paper, we extended DLDB in terms of these additional reasoning capabilities.

Instance Equalities. OWL does not make the unique names assumption, which means that different names do not necessarily imply different objects. Given that many individuals contribute to the Web, it is highly likely that different IDs will be used to refer to same object. Such IDs are said to be equal. A Semantic Web knowledge base system thus needs 1) IDs that are freely interchangeable and 2) an inference mechanism that actually treat them as one object. Usually, equality is encoded in OWL as *(a owl:sameAs b)*, where a and b are URIs.

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, especially if you consider that the number of tables is equal to the number of classes plus the number of properties). We devised some auxiliary tables to keep track of the tables that each id is appeared in. A table

`Individual_Occurrence(id, ontId, termId, position)`

is used to record the the occurrences of each id. Since this table is most likely to be the biggest table in the knowledge base, we have to be careful on the storage size of each row. All of the four fields are of type Integer. The `ontId` and `termId` are combined to indicate which class or property table an id has appeared, and the `position` is used to indicate which fields the id resides in case of a property table (1, 2 and 3 represent ‘subject’, ‘object’ and ‘both’ respectively). Another table

`Tables_Index(tableName, ontId, termId)`

is used to translate `ontId` and `termId` into an actual table name (string), so that the table names are stored only once in the database. To substitute an id with another id, the procedure queries the `Individual_Occurrence` table and `Tables_Index` table to find the set of tables upon which an update is issued to perform the substitution. Algorithm 1 illustrates the procedure of substituting an id with another one (usually the canonical id).

Algorithm 1 Substitute the original id with the target id

`SUBSTITUTEID(orig, targ)`

- 1: `ResultSet` \leftarrow query the `INDIVIDUAL_OCCURRENCE` table, find the tables and positions that *orig* occurs)
 - 2: **for** each tuple $\langle \textit{tableName}, \textit{position} \rangle \in \textit{ResultSet}$ **do**
 - 3: replace *orig* with *targ* in table *tableName* on position *pos*
 - 4: add the occurrence of *targ* in *tableName* on position *pos* to `INDIVIDUAL_OCCURRENCE` TABLE
 - 5: **end for**
 - 6: delete the occurrence of *orig* in `INDIVIDUAL_OCCURRENCE` TABLE
 - 7: update the `URI_INDEX` table, replace *orig* with *targ*
-

To decide which id should be used as canonical id, one can query the `individual_occurrence` table and find the most used id. This will minimize the number of update operations.

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. This happens after a certain number of triples have been loaded into the system. 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. The improved approach is shown in Algorithm 2.

Algorithm 2 Find instance equalities through a given table

```

FINDEQUALITIES(tableName)
1: ResultSet rs1, rs2;
2: Variable targ, orig;
3: rs1 ← EXECUTE-QUERY(
   SELECT object FROM tableName
   GROUP BY object HAVING COUNT (DISTINCT subject) > 1 )
4: for each tuple  $\langle a \rangle \in$  rs1 do
5:   rs2 ← EXECUTE-QUERY(
     SELECT subject FROM tableName WHERE object = a )
6:   for each tuple  $\langle b \rangle \in$  rs2 do
7:     if this is the tuple then
8:       targ ← b
9:     else
10:      orig ← b
11:     SUBSTITUTEID(orig, targ)
12:     end if
13:   end for
14: end for

```

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. Transitive closure is typically a problematic operation for an RDBMS. Nevertheless a number of algorithms have been proposed and some systems even support a built-in operation. In our system, 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 [10]. 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.

If we executed the SQL statement below whenever a tuple(s,o) is inserted to a table p1 that corresponds to a transitive property, then the table p1 maintains the transitive closure.

```
INSERT INTO p1 (  
SELECT s, o  
UNION  
SELECT s, p1.object FROM p1 WHERE p1.subject=o  
UNION  
SELECT p1.subject, o FROM p1 WHERE p1.object=s  
UNION  
SELECT tc1.subject, tc2.object FROM p1 as tc1, p1 as tc2  
WHERE tc1.object=s and tc2.subject=o);
```

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 will 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.

These cases can be solved by another algorithm, which is run periodically. This algorithm joins the table 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.

4.3 Indexing and Updates

We use indexes to improve the query performance. These indexes are independently maintained by the Hawkeye system without the intervention from database administrators. First, in addition to the primary keys, the system automatically indexes the object columns of property tables. This decision is based on the observation that many queries involve matching values for properties. Second, we note that there can be a load time penalty when looking up the URI-index table to find the ID for a URI. In order to reduce this overhead, we create indexes for both columns in the URI-index table. These indices significantly shorten the ID-lookup query time especially when the URI-index table is very long. Third, we index the ID column of *individual_occurrence* as well. Although adding indexes will increase the load time and the storage space, we think the tradeoffs are well rewarded by the improvements on query performance. In many cases, these queries are issued at load time and this potentially offsets the indexing and maintenance cost. In the future, we plan to quantitatively evaluate the impact of indexing.

To ensure the freshness of the data, our query answering system addresses some aspects of updates. The updates of a data document that is already in the system are

Algorithm 3 Compute transitive closure over a table

COMPUTETRANSITIVECLOSURE(*tableName*, *viewIndex*)

```
1: CREATE TEMPORARY TABLE temp
2: CREATE TEMPORARY TABLE delta
3: INSERT INTO temp
  SELECT p1.subject, p2.object, viewIndex, viewIndex
  FROM tableName:viewBy:viewIndex as p1, tableName:viewBy:viewIndex as p2
  WHERE p1.object = p2.subject;
4: INSERT INTO delta
  distinct records in temp but NOT in tableName:viewBy:viewIndex
5: repeat
6:   INSERT INTO temp
     SELECT p1.subject, p2.object, viewIndex, viewIndex
     FROM tableName:viewBy:viewIndex as p1, delta as p2
     WHERE p1.object = p2.subject;
7:   INSERT INTO temp
     SELECT p2.subject, p1.object, viewIndex, viewIndex
     FROM tableName:viewBy:viewIndex as p1, delta as p2
     WHERE p2.object = p1.subject;
8:   INSERT INTO tableName SELECT * FROM delta;
9:   DELETE FROM delta;
10:  INSERT INTO delta
     distinct records in temp but NOT in tableName:viewBy:viewIndex
11: until the number of rows in delta equals to zero
```

handled by deleting and reloading its individuals. However, the updates of ontologies are not currently supported. Ontologies are changed less frequently than data documents, and versioning policies may require a new location for updated ontologies. Our approaches of handling equalities and transitive properties make it difficult to retract the assertions involving these two features. We plan to investigate different solutions on this problem, one possibility is to use compensating transactions [20].

5 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 is of limited utility. 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. Several sources on the web can provide data for these scenarios, however, we use only some of them augment our Semantic Web data. The scenarios and the data sources used for the experiment are described below.

5.1 Scenarios

The first scenario involves an e-government domain. There is lot of information about government activities on the web. This information is about the representatives of the Congress and the Senate of United States (i.e. Congressmen and Senators), and about bills being sponsored by and voted for by various representatives in either houses. The power of the Semantic Web is best demonstrated when this data is combined with other sources. For example, using geographic and demographic information about United States in this scenario can facilitate queries such as: how many bills related to small businesses were sponsored by a politician who lives in a state with population less than a million; or which politicians of the Midwest region sponsored bills related to tornadoes.

In order to answer such queries using traditional search engines, one would have to manually navigate multiple data sources, gather them and align the data depending on the query (i.e. politicians according to regions and bills according politicians). The next step would be to manually narrow down the results by removing unwanted alignments by using criteria such regions or states with population less than a million or bills with words 'tornado' in them. It is obvious that this entire process is time-consuming and repetitive for each new query.

The second scenario that we use is an academic research domain. It involves people who are involved in research in universities such as professors and students, the publications written and cited by them, their co-authors and their co-workers. Various repositories of bibliographic information (e.g. DBLP, Citeseer, etc.) provide varied information about publications, authors and possibly citations. Citeseer provides citation information but indexes only online publications. DBLP is a more comprehensive repository, but it doesn't provide citation information.

To make the queries more realistic and interesting we use information about National Science Foundation(NSF) funds allocation, University data and the Artificial Intelligence Genealogy Project(AIGP) which provides the advisor-advisee relationships for many AI researchers. The above sources when integrated together allow us to answer queries such as: Find all papers written by academic descendants of Marvin Minsky who are at universities in the northeast U.S. and who have received NSF funding.

5.2 Data Sources

Table 1 describes the data sources used in our scenarios. The first column lists the shorthand prefixes we assigned to each data source. The second column lists the names of the data sources. The third column lists the original format of the data (and the ontology when available). The next two columns list some of the classes and properties defined in the ontology that are relevant to the queries that we chose for our experiment. The last column lists the number of RDF triples from each source.

The e-government scenario uses **g**¹, **c**², **b**³ and **n**⁴ data sources. Source **g** provides us with geographic information organized as a hierarchy of regions and states within United States. Source **c** provides us with basic biographic information about members of 107th congress in XML such as name, a government provided unique ID, party, congressional district and state. Source **b** provides us information about bills sponsored

in various congresses such as date, title, sponsor, votes for-against it, etc. It also provides the biographic information about the current and some previous congressmen and senators but to demonstrate integration, we rely on **c** to supply the biographic information. **n**, provides demographic census information for all the states such as population, households, land area and water area.

The academic publication scenario uses **d**⁵, **s**⁶, **a**⁷, **u**⁸, **w**⁹ and **f** (found from Swoogle's crawl) data sources. Source **s** refers to Citeseer, a repository which primarily focuses on publications in field of Computer Science. It provides citation information. Source **d** refers to DBLP, another repository with a much more comprehensive coverage but lacks the useful citation information. It has information about the papers published in this conference, authors, delegates, workshops, tutorials etc. Source **a** refers to data from the AI Genealogy project of the University of Texas. This project has information about doctorate computer science researchers involved in Artificial intelligence structured as advisor-student relations. Source **u** refers to University data, which provides information about various universities of United States such as name, website, etc. organized by the state. Source **w** refers to NSF Awards data, which gives information about NSF awards allocated to various research projects in universities across US. It includes principle investigator, date, university, etc. Many classes and properties in the transformed data which describe people, viz. Author, Researcher, etc. We use foaf:Person to describe them and hence facilitate the usage of large amounts of FOAF data available on the semantic web which conforms to the FOAF ontology. This data does not need any kind of transformation and is referred to as **f**.

To augment our Semantic Web data we transformed the data from these sources to RDF which commits to valid ontologies. The original format of **g** is a DAML ontology and data. We translated the ontology and data into OWL files by using simple find and replace techniques. Source **c**, **d** and **s** are originally in XML format. We developed an ontology for each of them based on their XML and developed domain specific scripts to translate the XML to RDF that conforms to ontologies we created. Source **b** is already in RDF; we created an ontology based on the data. For **n**, whose original format was N3, we created an ontology based on the data and a domain specific script to translate the data into RDF that conforms to the ontology. Source **a**, **u** and **w** are sets of HTML web pages. For each of these we developed scripts to collect the pages and extract the desired information from these pages. We developed ontologies for each of them and converted them into RDF.

¹ <http://www.daml.ri.cmu.edu/ont/USRegionState.daml>

² <http://xml.house.gov/Members/mbr107.xml>

³ <http://www.govtrack.us/data/rdf/>

⁴ <http://www.govtrack.us/data/rdf/census.tgz>

⁵ <http://dblp.uni-trier.de/xml/>

⁶ <http://citeseer.ist.psu.edu/oai.html>

⁷ <http://aigp.csres.utexas.edu/aigp/>

⁸ <http://en.wikipedia.org/wiki/List>

⁹ <http://www.nsf.gov/awardsearch/>

Pre	Data Source	Original Format	Classes	Properties	Triples
a	AIGP	No Ontology Set of HTML Pages	AIResearcher	hasAdvisor influencedBy hasInfluenced	5973
b	Bill Data	No Ontology RDF	Politician Bill	name sponsoredBy shortTitle	75711
c	107th Congress	No Ontology XML Data	Member USCD	party fromUSCD isIn	2628
d	DBLP	No Ontology XML Data	Article foaf:Person	author coauthor	15523209
f	FOAF	RDF Schema Ontology	Person	knows	11601453
g	Geographic Data	DAML Ontology DAML Data	USRegion USState	memberstate region	578
n	Census Data	No Ontology N3 File	State	population landarea	314
s	Citeseer	No Ontology XML Data	Article foaf:Person	author references coauthor	7630021
u	University Data	No Ontology Set of HTML Pages	University	state website name	16767
w	NSF Awards Data	No Ontology Set of HTML Pages	NSFAward	principalInvestigator state	462102

Table 1. Data sources Summary

5.3 Maps

The purpose of using data from multiple sources is that a single source doesn't hold all the information for a certain individual. For example, **c** gives us biographic information of congress members, while **b** gives us information on bills sponsored and voted by them. If information is provided by two different sources, then different URIs are likely to be used to identify the same individual. Sometimes, there are standard identification schemes for individuals, and mapping will not be necessary. For example, both **c** and **b** use unique seven character IDs provided by the government (such as 'A000014' for congressman Neil Abercrombie). In these sources, we add a standard prefix to form the URI such as 'http://www.house.gov/members#A000014'.

When different URIs are used, then we must explicitly annotate that they are equivalent using the owl:sameAs property. For example, the URI schemes of **d** and **s** are different. We generated files which explicitly declare that the individuals referred to by URIs in **d** and **s** are the same. We note that there are a number of techniques with varying accuracy for automatic coreference resolution [26]. However for simplicity, we match the names of individuals (e.g. Authors) from each of the sources using simple string matching techniques and conclude that the two URIs refer to the same individ-

ual if the names match. In such an event we generate an owl:sameAs relation for these URIs. We would expect that these sameAs mappings would typically be contributed by many individuals and found throughout the web. Totally, we created 204,062 sameAs statements between 4 pairs of data sources. Table 2 summarizes the number of sameAs statements that we generated for various data sources.

Source	Target	# of sameAs
s:author	d:author	199641
a:AIResearcher	d:author	1767
a:AIResearcher	w:PI	1224
u:University	w:PIOrg	1380
g:State	c:State	50
TOTAL		204,062

Table 2. Summary of sameAs Relations

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, the Member class of the congress ontology is equivalent to the Congressman class of the bill ontology. The property uscd of the congress ontology is equivalent to the congressmanOf property in the bill ontology. Similarly, 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 while the class AIResearcher of AIGP ontology is subClassOf the class *foaf:Person* that we use to define people in the Citeseer ontology. Table 3 describes the maps which we used in our experiment. The maps that we use for experiments are relatively simple, however one can envision more complex maps in different domains.

Maps for E-government Scenario	Maps for Academic Scenario
c:Member \equiv b:Congressman	e:Paper \sqsubseteq d:Publication
c:name \sqsubseteq b:name	e:Paper \sqsubseteq s:Publication
c:uscd \equiv b:congressmanOf	e:creator \sqsupseteq d:author
g:region \equiv c:isInRegion	e:creator \sqsupseteq s:author
g:region \sqsubseteq c:isIn	e:references \equiv e:isReferencedBy
g:USState \equiv c:State	a:AIResearcher \sqsubseteq f:Person

Table 3. Maps

5.4 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 SUN workstation featuring

dual 64-bit Opteron CPUs and 10GB main memory. The RDBMS is MySQL 5.0 and the DL reasoner is Racer Pro. It took 650 hours to process the 1.7 million urls that we got 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 166 million 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 4 shows the cumulative load time after each million triples load into the system. In general we see that the load time increases gradually and our system scales well. Note that the total time in this chart only includes the time to retrieve and process those documents that had been successfully downloaded. 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 *InverseFunctional-Property* 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.

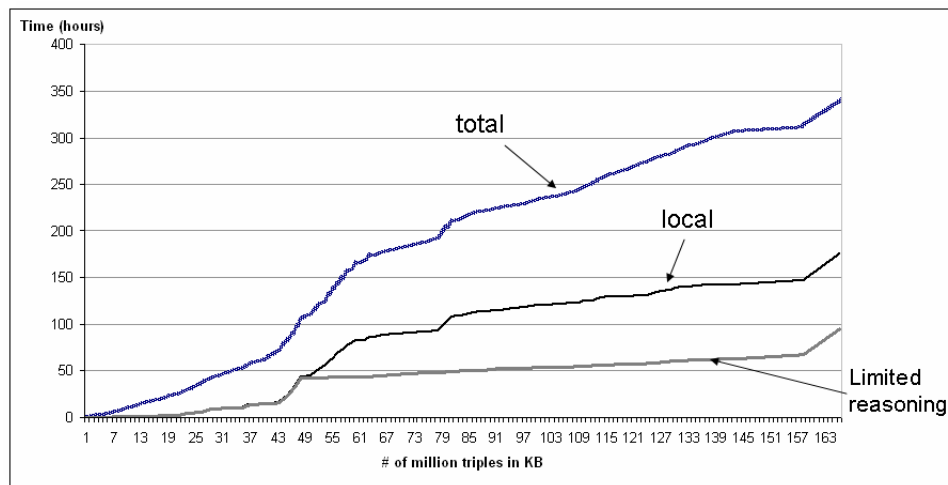


Fig. 4. Hawkeye Cumulative Load time

Query Performance. In order to evaluate the query performance of Hawkeye, we used the six query templates described in Table 4. For each query template, we listed the inferences and the data sources. Note the mappings between different sources are indispensable too. Table 5 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 note that although there is diversity in the results of the queries, the response time remains fairly stable. 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.

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 4. Query descriptions

We note that the Aca2 query takes about 6 minutes to complete. This is due to the fact that it uses the foaf:knows property. This property results in 11 million records in our database. In addition dblp:coauthor and citeseer:coauthor are both sub properties of foaf:knows and have additional 5 million records. 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 Aca2 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.

Query template	No. of variations	Response time (ms)		No. of results	
		Avg.	Stddev.	%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 5. Query performance

6 Related work

A large amount of current research in web source integration by the database community has focused on the so called deep web[6]. Deep web integration focuses on interaction integration rather than semantic integration. In that sense we view this research as orthogonal to ours. The Piazza system [28] and the SWIM middleware [8] present techniques of integrating XML data sources. Their maps are expressed in XQuery path expressions. As XQuery does not have a declarative logical semantics we can not use a general purpose reasoner to work on them. Second, their mapping language XQuery is different from the data they integrate (XML). Our maps relate an OWL concept with another OWL concept using axioms from a subset of OWL. This way we do not need additional processing such as XSLT transformation to realize the effect of our maps.

We claim that this is the first attempt to reason with the Semantic Web as a whole. Here we use reasoning strictly as a knowledge representation term. We should however note that there are several projects that process the Semantic Web in various other ways. For example, Swoogle [9] is the largest index of Semantic Web documents. However, Swoogle’s query and retrieval mechanism is basically an information retrieval system. The mechanism relies on string match and relevancy computation. This does not exploit the reasoning that can be done over Semantic Web data. Flink [21] is a presentation of the scientific work and social connectivity of Semantic Web researchers. This interesting application uses FOAF to develop social maps. The Carnot [17] system is one of the earliest works that uses articulation (i.e. mapping) axioms to address heterogeneity in enterprise integration. Carnot depends on commonsense knowledge and is more of a centralized approach which is not a suitable model for the Web.

There are some ongoing efforts to bootstrap the Semantic Web by providing ontologies and reusable knowledge bases. TAP [11] is one example of such effort. They provide a fairly comprehensive source of basic information about popular entities, like music, authors, autos etc. They do not replace the upper ontologies but complement them. Their work can be seen as “grounding” of some fragment of those upper ontologies with real world data. Bibster [4] is a Peer-to-Peer system for exchanging bibliographic data among researchers. Bibster exploits ontological information in data-storage, query formulation, query-routing and answer presentation. “CS AKTive Space” [27] is an application that exploits a wide range of heterogeneous and distributed content relating

to Computer Science research in the UK. The content currently comprises around ten million RDF triples. The content is mediated through an ontology constructed for the application domain and incorporates components from other published ontologies. Although these efforts have large amounts of data, they each assume a common ontology.

The C-OWL work [3] proposed that ontologies could be contextualized to represent local models from a view of a local domain. They suggested that each ontology is an independent local model. If some other ontologies' vocabularies need to be shared, some bridge rules should be appended to the ontology which extends those vocabularies. Compared to C-OWL, our perspective approach also provides multiple models from different views without modifying the current Semantic Web languages.

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. The system that most resembles the capabilities of DLDB is KAON2 [18], which uses a novel algorithm to reduce OWL DL into disjunctive datalog programs.

7 Conclusion and future work

In this paper we present DLDB, a system that uses ontology alignments expressed in OWL to provide a uniform view of the Semantic Web to the user. 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 200 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 defer integration until query time and thus provide a framework where the user is not bound by a predetermined schema (or alignment of schemas). 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, and any user with access to the Web has **automatic** access to all the maps. We believe this shared mapping approach will produce a network effect on the growth of mapped ontologies as the users establishing alignments for their own purpose in turn is contributing to the overall integration of the Semantic Web. Our system is optimized for working with OWL instances. 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. We have already identified some potential optimizations such as the automatic query rewriting. Second, although perspectives re-

solve some form of inconsistencies; inconsistency between data sources that commit to the same ontology still result in inconsistent perspectives. These are some interesting research questions that we plan to explore and invite others to do the same.

8 Acknowledgment

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

References

1. F. Baddier et al. *The Description Logic Handbook - Theory, Implementation and Applications*. Cambridge University Press, Cambridge, UK, 2003.
2. S. Bechhofer and R. Volz. Patching syntax in OWL ontologies. In *Proceedings of the Third International Semantic Web Conference, 2004*.
3. P. Bouquet, F. Giunchiglia, F. van Harmelen, L. Serafini, and H. Stuckenschmidt. C-OWL: Contextualizing ontologies. In *Proc. of the 2003 Int'l Semantic Web Conf. (ISWC 2003)*, LNCS 2870, pages 164–179. Springer, 2003.
4. J. Broekstra, M. Ehrig, P. Haase, F. Harmelen, M. Menken, P. Mika, B. Schnizler, and R. Siebes. Bibster - a semantics-based bibliographic peer-to-peer system. In *Proceedings of the International Semantic Web Conference (ISWC2004)*, 2004.
5. J. Broekstra, A. Kampman, and F. van Harmelen. Sesame: A generic architecture for storing and querying rdf and rdf schema. In *Proc. of the First International Semantic Web Conference*, 2002.
6. K. Chang, B. He, C. Li, and Z. Zhang. Structured databases on the web: Observations and implications, 2003.
7. N. Choi, I.-Y. Song, and H. Han. A survey on ontology mapping. *SIGMOD Rec.*, 35(3):34–41, 2006.
8. V. Christophides, G. Karvounarakis, A. Magkanaraki, D. Plexousakis, et al. The ics-forth semantic web integration middleware (swim). In *Proceedings of the First International Workshop on Semantic Web and Databases 2003 (SWDB 2003)*, pages 381–393, 2003.
9. 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.
10. G. Dong, L. Libkin, J. Su, and L. Wong. Maintaining transitive closure of graphs in SQL. *Int. Journal of Information Technology*, 1999.
11. R. Guha. Tap: Towards the semantic web. Demo on World Wide Web 2002 Conference, 2002. At: <http://tap.stanford.edu/www2002.ppt>.
12. Y. Guo, Z. Pan, and J. Heflin. LUBM: A benchmark for owl knowledge base systems. *Journal of Web Semantics*, 3(2):158–182, 2005.
13. Y. Guo, A. Qasem, Z. Pan, and J. Heflin. A requirements driven framework for benchmarking semantic web knowledge base systems. *IEEE Transactions on Knowledge and Data Engineering*, 19(2):297–309, Feb 2007.
14. A. Halevy, M. Franklin, and D. Maier. Principles of dataspace systems. In *PODS '06: Proceedings of the twenty-fifth ACM SIGMOD-SIGACT-SIGART symposium on Principles of database systems*, pages 1–9, New York, NY, USA, 2006. ACM Press.
15. 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.

16. I. Horrocks and S. Tessaris. A conjunctive query language for description logic aboxes. In *AAAI/IAAI*, pages 399–404, 2000.
17. M. Huhns, N. Jacobs, T. Ksiezyk, W. M. Shen, M. Singh, and P. Canata. Enterprise information modeling and model integration in carnot. In *Proc. of the International Conference on Intelligent and Cooperative Information Systems*, pages 12–14, 1993.
18. 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.
19. J. Kopena and W. Regli. DAMLJessKB: A tool for reasoning with the semantic web. *IEEE Intelligent Systems*, 18(3):74–77, May/June 2003.
20. H. F. Korth, E. Levy, and A. Silberschatz. A formal approach to recovery by compensating transactions. In *Proceedings of the sixteenth international conference on Very large databases*, pages 95–106, San Francisco, CA, USA, 1990. Morgan Kaufmann Publishers Inc.
21. P. Mika. Flink: Semantic web technology for the extraction and analysis of social networks. *Journal of Web Semantics*, 3(2), 2005.
22. Z. Pan. Hawk descriptions, 2005. <http://swat.cse.lehigh.edu/downloads/hawk.html>.
23. 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 Systems, ISWC*, 2003.
24. 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.
25. E. Prud'hommeaux and A. Seaborne. SPARQL query language for rdf. TR, October 2006. <http://www.w3.org/TR/rdf-sparql-query/>.
26. P. Reuther and B. Walter. Survey on test collections and techniques for personal name matching. *International Journal of Metadata, Semantics and Ontologies*, 1(2):89–99, 2006.
27. 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*, pages 384–392, New York, NY, USA, 2004. ACM Press.
28. I. Tatarinov, Z. Ives, J. Madhavan, A. Halevy, D. Suciu, N. Dalvi, X. L. Dong, Y. Kadiyska, G. Miklau, and P. Mork. The Piazza peer data management project. *SIGMOD Rec.*, 2003.
29. Y. Theoharis, V. Christophides, and G. Karvounarakis. Benchmarking database representations of rdf/s stores. In *Proc. of the 4th International Semantic Web Conference*, 2005.
30. W3C. SKOS: Simple knowledge organization for the web. Activity, 2006. <http://www.w3.org/2004/02/skos/>.
31. 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.
32. W. E. Winkler. Data cleaning methods. In *Proceedings of the 2003 ACM SIGKDD Workshop on Data Cleaning, Record Linkage, and Object Consolidation*, pages 1–6, Washington, DC, 2003.
33. M.-C. Wu and A. P. Buchmann. Research issues in data warehousing. In *Datenbanksysteme in Büro, Technik und Wissenschaft*, pages 61–82, 1997.