Benchmarking DAML+OIL Repositories

Yuanbo Guo, Jeff Heflin, and Zhengxiang Pan

Department of Computer Science and Engineering, Lehigh University
19 Memorial Drive West, Bethlehem, PA 18015
{yug2,heflin,zhp2}@cse.lehigh.edu

Abstract. We present a benchmark that facilitates the evaluation of DAML+OIL repositories in a standard and systematic way. This benchmark is intended to evaluate the performance of DAML+OIL repositories with respect to extensional queries over a large data set that commits to a single realistic ontology. It consists of the ontology, customizable synthetic data, a set of test queries, and several performance metrics. Main features of the benchmark include a plausible ontology for the university domain, a repeatable data set that can be scaled to an arbitrary size, and an approach for measuring the degree to which a repository returns complete query answers. We also show a benchmark experiment for the evaluation of DLDB, a DAML+OIL repository that extends a relational database management system with description logic inference capabilities.

1 Introduction

DAML+OIL [6], built on RDF and RDF Schema [23] as a markup language for Web resources, is the subject of much research. One important issue in using DAML+OIL is to explore efficient and scalable mechanisms for storing and querying its semantic data. A variety of systems may be used as the repository, for example, relational databases, simple knowledge representation systems, deductive databases, and description logic systems. Furthermore, given a particular style of system such as relational database, numerous schemata are possible [1, 2, 11, 19].

Another issue, which is closely related to the previous one, is to exploit the semantics of DAML+OIL. DAML+OIL extends RDF and RDF Schema with richer modelling primitives and has a clear and well defined semantics [6]. Hence techniques like ontology reasoning could be integrated into the repositories in order to enhance them.

Clearly every approach has its pros and cons. Given so many potential combinations of systems, schemata and techniques for constructing DAML+OIL repositories, there is obviously a need for a standard and systematic way to evaluate those repositories in order to help users to choose an implementation that meets their needs.

This paper presents a first step to meet such a need. We have developed the Lehigh University benchmark (LUBM) to facilitate the evaluation of DAML+OIL repositories. The benchmark consists of a realistic ontology, customizable synthetic test data, a set of test queries, and several performance metrics. In order to use the

D. Fensel et al. (Eds.): ISWC 2003, LNCS 2870, pp. 613–627, 2003.

benchmark, we have also implemented related facilities. We show in the paper how we designed the benchmark and tailored it to the Semantic Web and DAML+OIL. In addition, based on the benchmark, we have successfully conducted an experiment to evaluate DLDB [20], a DAML+OIL repository developed by us, which extends a relational DBMS with description logic inference capabilities.

The rest of the paper is organized as follows: Section 2 details the design of the benchmark and talks about some implementation issues as well. Section 3 presents the aforementioned experiment. Section 4 discusses some related work. Section 5 concludes.

2 The Benchmark

As noted, the goal of the Lehigh University benchmark is to standardize and facilitate the evaluation of repositories for DAML+OIL. It has four major components: a benchmark ontology, benchmark data, test queries, and performance metrics. It also consists of a tool for synthetic data generation. We describe them in the following sections.

2.1 Benchmark Ontology

Recognizing that on the Semantic Web, data will by far outnumber ontologies, we wanted to develop a benchmark for this. Therefore, we chose to generate large amounts of data for a single ontology of moderate size. We aimed for this data to be as realistic as possible.

We have created the Univ-Bench ontology [13] as the benchmark ontology, based on which the benchmark data are generated. The ontology describes universities and departments and the activities that occur at them. It currently contains 43 classes and 32 properties. Its predecessor is the Univ1.0 ontology [12], which has been used to describe DAML data about actual universities and departments. We chose this ontology expecting that its domain would be familiar to most of the benchmark users. In creating the benchmark ontology, we have made some changes to the Univ1.0 ontology mainly to use more DAML+OIL features that are required by the benchmark. For instance, the Univ1.0 ontology states that GraduateStudent is a subclass of Student. In the Univ-Bench ontology, we have replaced that definition with what is shown in Fig. 1, using restriction. As a result, the subclass relationship between the two classes must be inferred using DAML+OIL semantics.

2.2 Benchmark Data

The benchmark data are the DAML+OIL data to be stored to the target repository. We have adopted a method of synthetic data generation. This serves multiple purposes. As with the Wisconsin benchmark [3, 4], a standard and widely used database benchmark, this allows us to control the selectivity and output size of each test query. However, there are some other DAML+OIL specific considerations:

```
<daml:Class ID="GraduateStudent">
    <rdfs:label>graduate student</rdfs:label>
    <daml:subClassOf>
        <rdfs:Class>
            <daml:intersectionOf rdf:parseType="daml:collection">
                <daml:Class rdf:about="#Person" />
                <daml:Restriction>
                    <daml:onProperty rdf:resource="#takesCourse" />
                    <daml:hasClass>
                        <daml:Class rdf:about="#GraduateCourse" />
                    </daml:hasClass>
                </daml:Restriction>
            </daml:intersectionOf>
        </rdfs:Class>
    </daml:subClassOf>
</daml:Class>
<daml:Class ID="Student">
    <rdfs:label>student</rdfs:label>
    <daml:sameClassAs>
        <rdfs:Class>
            <daml:intersectionOf rdf:parseType="daml:collection">
                <daml:Class rdf:about="#Person" />
                <daml:Restriction>
                    <daml:onProperty rdf:resource="#takesCourse" />
                    <daml:hasClass>
                        <daml:Class rdf:about="#Course" />
                    </daml:hasClass>
                </daml:Restriction>
            </daml:intersectionOf>
        </rdfs:Class>
    </daml:sameClassAs>
</daml:Class>
<daml:Class ID="GraduateCourse">
    <rdfs:label>Graduate Level Courses</rdfs:label>
    <rdfs:subClassOf rdf:resource="#Course" />
</daml:Class>
```

Fig. 1. Definition of classes Student and GraduateStudent

1. We would like the benchmark data to be of a range of sizes including considerably large ones. It is hard to find such data sources that are based on the same ontology.
2. We may need the presence of certain kinds of instances in the benchmark data. This allows us to design repeatable tests for as many representative query types as possible. These tests not only evaluate the storage mechanisms for DAML+OIL data but also the techniques that exploit DAML+OIL semantics. We may rely on instances of certain classes and/or properties to test against those techniques.

Data generation is carried out by UBA (Univ-Bench Artificial data generator), a tool we have developed for the benchmark. UBA features random and repeatable generation of DAML+OIL instances. A university is the minimum unit of data generation and for each university, a set of files describing its departments are generated. Instances of both classes and properties are randomly decided. To make the data as realistic as possible, some restrictions are applied based on common sense and domain investigation. Examples are ”a minimum of 15 and a maximum of 25 departments in each university”, ”an undergraduate student/faculty ratio between 8

and 14 inclusively", "each graduate student takes at least 1 but at most 3 courses", and so on and so forth.

UBA identifies universities by assigning them zero-based indexes, e.g., the first university is named University0. Data generated by the tool are exactly repeatable in respect of universities. This is possible because the tool allows the user to enter an initial seed for the random number generator that is used in the data generation process.

UBA makes it possible to systematically produce the benchmark data. We may specify how many and which universities to generate. And information such as the number of instances of each class and property is provided so that we can easily determine how many universities to load to the repository in order to conduct the experiment on data of a specific size. In the benchmark, we choose data sets via looking at the total number of class instances. To help identify the data set, we introduce the following notation:

LUBM(N, S): The data set that contains N's universities beginning at University0 and is generated by the UBA tool starting with seed S.

We use this notation in section 3 when describing the experiment.

2.3 Test Queries

We have constructed thirteen queries for the benchmark, as shown at the end of the section. For each query, we first describe it in natural language and then in a KIF [10] like language, in which a query is written as a conjunction of atoms.[1] Then we analyze the characteristics of the query to show why it has been chosen.

In choosing the queries, first of all, we wanted them to be realistic. Meanwhile, we have mainly taken into account following factors:

1. *Input size.* This is measured as the proportion of the class instances involved in the query to the total class instances in the benchmark data. Here we refer to not just class instances explicitly expressed but also those that are entailed by the knowledge base. We define the input size as large if the proportion is greater than 5%, and small otherwise.
2. *Selectivity.* This is measured as the estimated proportion of the class instances involved in the query that satisfy the query criteria. We regard the selectivity as high if the proportion is lower than 10%, and low otherwise. Generally, we did the estimation based on DAML+OIL semantics, the Univ-Bench ontology, the benchmark data model, and the actual data sets. Whether the selectivity is high or low for a query may depend on the data set used. For instance, the selectivity of Queries 8, 11 and 12 is low if the data set contains only University0 while high if the data set contains more than 10 universities.
3. *Complexity.* We use the number of classes and properties that are involved in the query as an indication of complexity. Since we do not assume any specific implementation of the repository, the real degree of complexity may vary by systems and schemata. For example, in a relational database the number may

[1] As we will show later, the benchmark user is not restricted to use this query language. We use it here for the sake of the subsequent explanation and also because we use it in the experiment.

directly indicate the times of join, which is a significant operation, or may not depending on the schema design.

4. *Hierarchy information assumed.* This considers whether information of class hierarchy or property hierarchy is required to achieve the complete answer. (We define completeness in next section).
5. DAML+OIL inference *assumed.* This considers whether DAML+OIL inference is required to achieve the completeness of the answer. DAML+OIL features used in the test queries include subsumption, i.e., inference of implicit subclass relationship, TransitiveProperty, inverseOf, and realization, i.e., inference of the most specific concepts that an individual is an instance of. One thing to note is that we are not benchmarking complex description logic reasoning. We are concerned with extensional queries. Some queries use simple description logic reasoning mainly to verify that this capability is present.

We have chosen test queries that cover a range of types in terms of the above criteria. At the same time, to the end of performance evaluation, we have emphasized queries with large input and high selectivity. If not otherwise noted, all the test queries are of this type. Some subtler factors have also been considered in designing the queries, such as the depth and width of class hierarchies[2], and the way the classes and properties chain together in the query.

Query1: All the graduate students who take GraduateCourse0 at Department0 of University0.

(type GraduateStudent ?X)
(takesCourse ?X http://www.Department0.University0.edu/GraduateCourse0)

This query bears large input and high selectivity. It is the simplest in the query set: it queries about just one class and one property and does not assume any hierarchy information or DAML+OIL inference.

Query2: All the graduate students who are now studying at the university from which they obtained their bachelor's degrees.

(type GraduateStudent ?X)
(type University ?Y)
(type Department ?Z)
(memberOf ?X ?Z)
(subOrganizationOf ?Z ?Y)
(undergraduateDegreeFrom ?X ?Y)

This query increases in complexity: 3 classes and 3 properties are involved. Additionally, there is a triangular pattern of relationships between the objects involved.

Query3: All the publications of AssistantProfessor0 at Department0 of University0.

(type Publication ?X)
(publicationAuthor ?X
http://www.Department0.University0.edu/AssistantProfessor0)

This query is similar to Query 1 but class Publication has a wide hierarchy.

[2] We define a class hierarchy as deep if its depth is greater than 3, and as wide if its average branching factor is greater than 3.

Query4: All the professors at Department0 of University0 and their email addresses and telephone numbers

(type Professor ?X)
(worksFor ?X http://www.Department0.University0.edu)
(name ?X ?Y1)
(emailAddress ?X ?Y2)
(telephone ?X ?Y3)

This query has small input and high selectivity. It assumes subClassOf relationship between Professor and its subclasses. Class Professor has a wide hierarchy. Another feature is that it queries about multiple properties of a single class.

Query5: All the members of Department0 of University0.

(type Person ?X)
(memberOf ?X http://www.Department0.University0.edu)

This query assumes subClassOf relationship between Person and its subclasses and subPropertyOf relationship between memberOf and its subproperties. Moreover, class Person features a deep and wide hierarchy.

Query6: All the students

(type Student ?X)

This query concerns only one class, but it assumes both the explicit subClassOf relationship between UndergraduateStudent and Student and the implicit one between GraduateStudent and Student. In addition, this query represents those with large input and low selectivity.

Query7: All the students who take the courses of AssociateProfessor0 at Department0 of University0.

(type Student ?X)
(type Course ?Y)
(teacherOf http://www.Department0.University0.edu/AssociateProfessor0 ?Y)
(takesCourse ?X ?Y)

This query is similar to Query 6 in terms of class Student but it increases in the number of classes and properties and its selectivity is high.

Query8: All the students of University0 and their email addresses.

(type Student ?X)
(type Department ?Y)
(memberOf ?X ?Y)
(subOrganizationOf ?Y http://www.University0.edu)
(emailAddress ?X ?Z)

This query is further more complex than Query 7 by including one more property. And as mentioned, the selectivity of this query and also Queries 11 and 12 depends on the data set.

Query9: All the students who take the courses of their advisors.

(type Student ?X)
(type Faculty ?Y)
(type Course ?Z)
(advisor ?X ?Y)
(takesCourse ?X ?Z)
(teacherOf ?Y ?Z)

Besides the aforementioned features of class Student and the wide hierarchy of class Faculty, this query is characterized by the most classes and properties in the query set. Like Query 2, there is a triangular pattern of relationships.

Query10: All the students who take GraduateCourse0 at Department0 of University0.
(type Student ?X)
(takesCourse ?X http://www.Department0.University0.edu/GraduateCourse0)
This query differs from Queries 6, 7, 8 and 9 in that it only requires the (implicit) subClassOf relationship between GraduateStudent and Student, i.e., subClassOf relationship between UndergraduateStudent and Student does not add to the results.

Query11: All the research groups at University0.
(type ResearchGroup ?X)
(subOrganizationOf ?X http://www.University0.edu)
Queries 11, 12 and 13 are intended to verify the presence of certain DAML+OIL reasoning capabilities in the repository. In this query, property subOrganizationOf is defined as transitive. Since in the benchmark data, instances of ResearchGroup are stated as a sub-organization of a Department individual and the later sub-organization of a University individual, inference about the subOrgnizationOf relationship between instances of ResearchGroup and University is required to answer this query. Additionally, input of this query is small.

Query12: All the department chairs of University0.
(type Chair ?X)
(type Department ?Y)
(worksFor ?X ?Y)
(subOrganizationOf ?Y http://www.University0.edu)
The benchmark data do not produce any instances of class Chair. Instead, each Department individual is linked to the chair professor of that department by property head. Hence this query requires realization, i.e., inference that that professor is an instance of class Chair because he or she is the head of a department. Input of this query is small as well.

Query13: All the alumni of University0.
(type Person ?X)
(hasAlumnus http://www.University0.edu ?X)
Property hasAlumnus is defined in the benchmark ontology as the inverse of property degreeFrom, which has three subproperties:
undergraduateDegreeFrom, mastersDegreeFrom, and doctoralDegreeFrom.
The benchmark data state a person as an alumnus of a university using one of these three subproperties instead of hasAlumnus. Therefore, this query assumes subPropertyOf relationships between degreeFrom and its subproperties, and also requires inference about inverseOf.

2.4 Performance Metrics

We have introduced four performance metrics for the evaluation of DAML+OIL repositories, defined as follows:

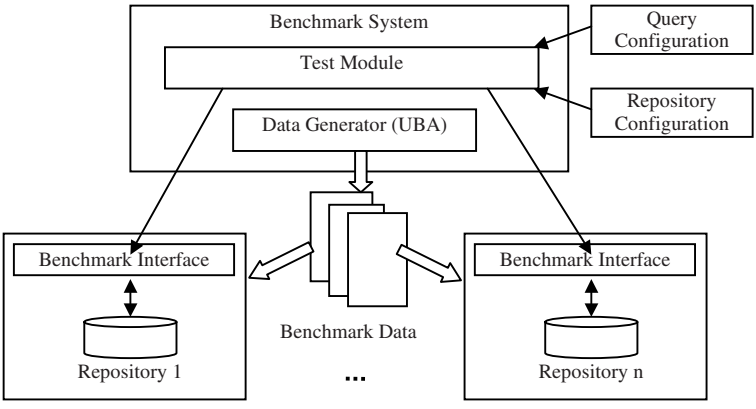

Fig. 2. Architecture of the Benchmark System

1. *Load time.* The stand alone elapsed time for storing the benchmark data to the repository. This also counts the time spent in any processing of the ontology and source files, such as parsing and reasoning.
2. *Repository size.* The consequent size of the repository after loading the specified benchmark data into it.
3. *Query response time.* The stand alone elapsed time of the query. The benchmark measures the time of each query on the target repositories as the following:
   For each target repository:
      For each test query:
         Open the repository.
         Execute the query on the repository consecutively for 10 times and compute the average response time. Each time:
            Issue the query, obtain the pointer to the result set, traverse that set sequentially, and collect the elapsed time.
         Close the repository.
4. *Completeness.* In logic, an inference procedure is complete if it can find a proof for any sentence that is entailed by the knowledge base. With respect to queries, we say a repository is complete if it generates all answers that are entailed by the knowledge base, where each answer is a binding of the query variables that results in an entailed sentence. However, on the Semantic Web, partial answers will often be acceptable. So it is important not to measure completeness with such a coarse distinction. Instead we measure degree of completeness of each query answer as the percentage of the complete results.

Among the above metrics, while the first three are standard (query response time was introduced in the Wisconsin benchmark [3, 4], and load time and repository size have been commonly used in other database benchmarks, e.g., the OO1 benchmark [5]), completeness is a new metric we developed for our benchmark.

2.5 Implementing the Benchmark on a Repository

Fig. 2 depicts the architecture of the benchmark system. The benchmark prescribes an interface to be implemented by each target repository. Through the interface, the benchmark system launches the loading process, requests operations on the repository (e.g. open and close), issues queries and obtains the results.

The benchmark user makes the benchmark system aware of the repositories by declaring them in a repository configuration file. The test queries are materialized via a specific language in a query configuration file. The language used depends on the functionality of the repository and the objective of the benchmarking task. The benchmark system just reads the lines of each query from the configuration file and passes them to each repository.

3 An Experiment Using the Benchmark

3.1 Introduction

DLDB [20] is a repository we have developed for the processing, storing and querying of DAML+OIL data. Its major feature is the extension of a relational database system with description logic inference capabilities. Specifically, DLDB uses Microsoft Access as the DBMS and FaCT [14, 15] as the DAML+OIL reasoner. It uses a database schema in which each class and property has its own table. Furthermore, views are employed to store hierarchy information of classes and properties. The view of each class is the union of its table and the views of all of its subclasses. Views of properties are implemented in a similar way by making use of the subproperty relationship.

The goal of the experiment is to evaluate DLDB and particularly the tradeoff between logical completeness and system performance. To this end, we have created two limited versions of DLDB and compared DLDB to them by using the benchmark. Therefore there are three repositories in the experiment, as follows:

1. *DLDB.* The aforementioned DLDB system.
2. *DLDB with No Reasoner (NR hereafter).* Same as DLDB except that the FaCT reasoning is not applied during loading the benchmark data. Thus the system does not have any implicit class hierarchy information.
3. *Plain.* A further simplified version of NR, wherein views are not used. Therefore the system does not explicitly use hierarchy information for classes or properties.

Test queries were shown in Section 2.3. The aforementioned benchmark interface was implemented in the three repositories respectively. Five different sizes of data sets containing 1, 5, 10, 20 and 50 universities were tested on those repositories. The experiment environment was the following:

Windows XP Professional Operating System;
1.80GHz Pentium 4 CPU;
256MB of RAM; 40GB of Hard Disk;
MS Access 2002 DBMS; JDBC/ODBC

We show the experiment results in next section.

Table 1. Load Time and Repository Sizes

Repository	Data Set	No. of Class Instances	Load Time (hh:mm:ss)	Repository Size (KB)
DLDB	LUBM(1, 0)	17,150	00:06:51	13,042
NR			00:06:46	13,042
Plain			00:06:46	12,943
DLDB	LUBM(5, 0)	107,421	00:42:39	73,925
NR			00:41:57	73,925
Plain			00:41:34	73,794
DLDB	LUBM(10, 0)	218,690	01:27:24	147,948
NR			01:25:32	147,948
Plain			01:26:02	147,816
DLDB	LUBM(20, 0)	462,316	03:06:32	311,099
NR			03:02:41	311,067
Plain			03:04:45	310,936
DLDB	LUBM(50, 0)	1,146,186	07:59:46	766,738
NR			07:49:40	766,738
Plain			07:52:19	766,607

3.2 Results and Analysis

Table 1 lists the time for loading the benchmark data to each repository and the consequent database sizes. To make them more comparable, we measured the database sizes after compacting the databases via using the MS Access database compacting utility, which rearranges how the database file is stored on the disk in order to improve the disk space efficiency.

Even though DLDB spends extra time in ontology reasoning, as displayed in the results there is no significant increase in its load time. This is unsurprising since DLDB only spends a fixed time doing reasoning at the start of loading so that the overhead is trivial and is quickly dominated by the time required to parse and load large amounts of data.

Overall, DLDB and NR have larger database sizes than Plain. This can be ascribed to the space overhead introduced by the use of views and the hierarchy information stored in the former two repositories. However the differences are minor: they are less than 1% for all the data sets. Moreover, it turns out that the overhead is stable as the data set size increases.

Table 2 shows the results of each test query on the three repositories, including the query response time, the number of answers and the answer completeness in percentage.

With respect to the completeness of answers, there are four patterns in the results, as described below. Meanwhile, we analyze the query response time inside each pattern and show some repesentative results via charts in Fig. 3.

1. The three repositories return the same number of answers. This happens when answering the query requires neither hierarchy information nor DAML+OIL inference. Queries 1, 2 and 3 belong to this pattern. As for the response time, the three repositories have close performance for Query 1. But as the chart of Query 2 shows, DLDB and NR takes significant longer time than Plain for the other two queries. This is not hard to understand if we refer to Section 2.3. Class Publication

Table 2. Results of Test Queries

Query	Repository & Data Set / Metrics	LUBM(1, 0)			LUBM(5, 0)			LUBM(10, 0)			LUBM(20, 0)			LUBM(50, 0)		
		DLDB	NR	Plain	DLDB	NR	Plain	DLDB	NR	Plain	DLDB	NR	Plain	DLDB	NR	Plain
1	Time(ms)	32	48	43	200	201	209	396	396	420	881	885	885	2295	2106	2248
	Answers	2	2	2	2	2	2	2	2	2	2	2	2	2	2	2
	Completeness	100	100	100	100	100	100	100	100	100	100	100	100	100	100	100
2	Time(ms)	139	132	20	850	867	142	1823	1871	382	5020	5060	1132	17782	16635	3854
	Answers	0	0	0	9	9	9	28	28	28	59	59	59	130	130	130
	Completeness	100	100	100	100	100	100	100	100	100	100	100	100	100	100	100
3	Time(ms)	168	182	31	2545	2506	115	5535	5518	215	11948	11867	437	35334	35157	1129
	Answers	6	6	6	6	6	6	6	6	6	6	6	6	6	6	6
	Completeness	100	100	100	100	100	100	100	100	100	100	100	100	100	100	100
4	Time(ms)	281	268	7	1996	1981	9	5159	5243	15	12740	12562	21	23100	21440	76
	Answers	34	34	0	34	34	0	34	34	0	34	34	0	34	34	0
	Completeness	100	100	0	100	100	0	100	100	0	100	100	0	100	100	0
5	Time(ms)	270	306	3	2729	3734	3	6348	8639	3	14615	19306	3	44428	56970	3
	Answers	719	719	0	719	719	0	719	719	0	719	719	0	719	719	0
	Completeness	100	100	0	100	100	0	100	100	0	100	100	0	100	100	0
6	Time(ms)	85	50	3	1073	526	4	2332	1075	3	5151	2346	7	14590	6553	7
	Answers	7790	5916	0	48582	36682	0	99566	75547	0	210603	160120	0	519842	393730	0
	Completeness	100	76	0	100	76	0	100	76	0	100	76	0	100	76	0
7	Time(ms)	351	68	17	1398	817	57	3126	1806	120	7082	4087	254	20134	11334	340
	Answers	34	30	0	34	30	0	34	30	0	34	30	0	34	30	0
	Completeness	100	88	0	100	88	0	100	88	0	100	88	0	100	88	0
8	Time(ms)	418	360	25	2843	2139	26	5982	4756	20	14031	11829	34	39250	32120	98
	Answers	7790	5916	0	7790	5916	0	7790	5916	0	7790	5916	0	7790	5916	0
	Completeness	100	76	0	100	76	0	100	76	0	100	76	0	100	76	0
9	Time(ms)	298	239	6	4962	3906	10	11953	8857	18	32948	24845	23	115782	41604	54
	Answers	113	57	0	715	353	0	1345	671	0	2864	1421	0	7368	3587	0
	Completeness	100	50	0	100	49	0	100	50	0	100	50	0	100	49	0
10	Time(ms)	92	48	3	1031	512	3	2264	1073	3	5000	2501	3	14525	6776	11
	Answers	2	0	0	2	0	0	2	0	0	2	0	0	2	0	0
	Completeness	100	0	0	100	0	0	100	0	0	100	0	0	100	0	0
11	Time(ms)	7	7	6	12	12	10	18	21	15	32	23	18	73	59	36
	Answers	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
	Completeness	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
12	Time(ms)	14	15	4	37	36	4	67	68	4	262	264	4	600	600	7
	Answers	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
	Completeness	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
13	Time(ms)	156	201	3	1792	2882	1	3903	6298	3	9185	14056	4	26846	41360	4
	Answers	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
	Completeness	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

in Query 3 has a wide hierarchy, and DLDB and NR are implemented to search all the way along the hierarchy even though that will not add any answers since all publications in the benchmark data are of class Publication. In Query 2, though only memberOf has one subproperty, due to the complexity of the query, i.e., it has five more classes and properties, even extra consideration of just one subproperty in combination with those classes and properties can lead to significant time overhead. In this case, the extra work by DLDB and NR will not add any answers either since all relationships between GraduateStudent and Department in the benchmark data are of property memberOf.

2. DLDB and NR return the same number of answers while Plain returns no answers. Queries 4 and 5 exhibit this pattern, both of which require class hierarchy information. It is not surprising that the former two repositories have much longer response time than Plain. Chart of Query 4 is shown in Fig. 3.

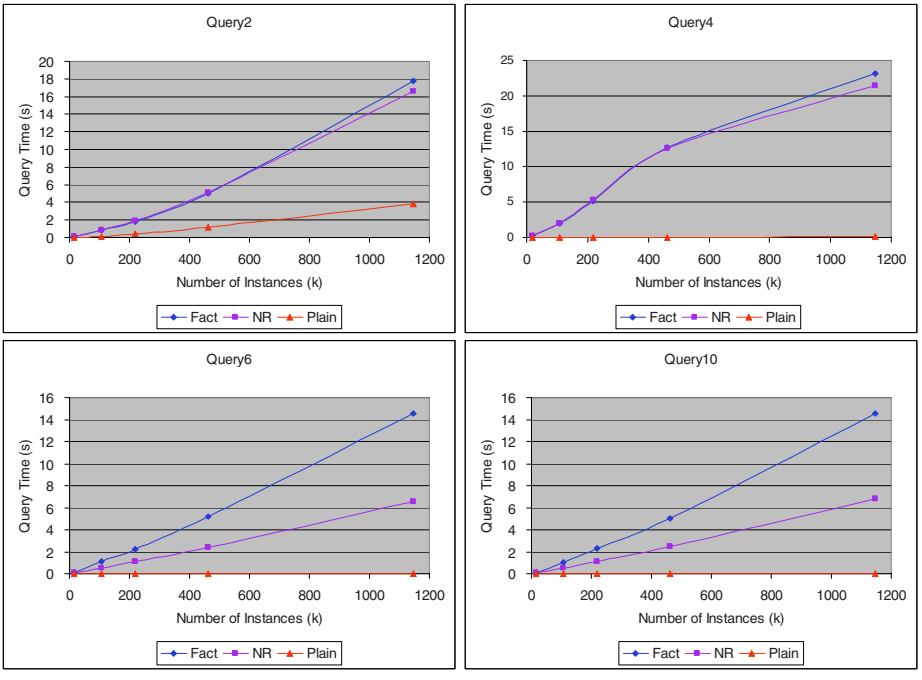

Fig. 3. Response Time for Queries 2, 4, 6 and 10

3. Both DLDB and NR return more than one answer while Plain returns no answers. In addition, DLDB returns more answers than NR. Queries of this kind include Queries 6, 7, 8 and 9. All of them require not only the hierarchy information but also DAML+OIL inference, which causes the difference among the three repositories. Response time for query 6 is shown in Fig. 3. We regard that the longer time DLDB takes than NR can mainly be ascribed to the larger answers it finds and handles.
4. DLDB returns multiple answers while both NR and Plain return no answers. Query 10 is the example, which requires inference capability to find any answer. Query 10's chart appears in Fig. 3. Again it is unsurprising that it costs DLDB longer time to handle more results than the others. In addition, it is notable that NR takes about half as long as DLDB to get no results.
5. None of the repositories is able to return any answers. Those queries include Queries 11, 12 and 13, which, as Section 2.3 shows, require certain DAML+OIL inferences that DLDB's reasoner is currently incapable of.

As far as the primary goal of this experiment is concerned, by the comparision of the three repositories, we see that the storage mechanism and the additional inference capabilities of DLDB have enabled it to achieve a considerably higher degree of the completeness of answers to many queries. Though this has to be traded off with performance mainly represented by the query response time, both for queries that can benifit from it and those that cannot, overall we consider the loss is small enough compared to what could potentially be gained.

More intensive analysis is possible when taking into account other charateristics of the queries that are described in section 2.3. In addition, the experiment results have hinted at some potential subjects. For one example, the curves for the queries as in Fig. 3 are all close to linear, suggesting the repositories are likely to scale. This motivates us to examine the issue with even larger data sets. For another, response time of NR for Queries 5 and 13 is outstandingly longer than that of DLDB, although we regard the two repositories are very similiar with respect to those two queries.[3] Also as shown above, even DLDB is unable to answer some queries with its current inference capability.

4 Related Work

To the best of our knowledge, the benchmark presented herein is the first one for DAML+OIL repositories in the Semantic Web area. There is a research work in benchmarking RDF schemata, which performs statistical analysis about the size and morphology of RDF schemata [18]. However this work does not provide a benchmark for evaluating a repository. [2] has developed some benchmark queries for RDF, however, these are mostly intensional queries, while we are concerned with extensional queries for DAML+OIL.

We have referred to several database benchmarks, including the Wisconsin benchmark [3, 4], the OO1 benchmark [5], and the SEQUOIA 2000 benchmark [22]. They are all DBMS-oriented benchmarks and storage benchmarks (vs. visualization benchmarks). The Lehigh University benchmark shares in spirit with them methodology and rationale in terms of the use of synthetic data, some criteria for choosing the test queries, and all but one of the metrics. However, our benchmark is tailored to the evaluation of DAML+OIL repositories and thus has many unique features. Particularly as shown in the previous sections, the benchmark contains an ontology, data sets, test queries and criteria that reflect special concepts, structures and concerns in the Semantic Web area such as classes and properties, logical completeness vs. system performance, etc. Moreover, our benchmark is intended to work with any DAML+OIL repositories, not just database systems.

Some attempts have been done to benchmark description logic systems [8, 16]. The emphasis of this work is to evaluate the reasoning algorithms in terms of the tradeoff between expressiveness and tractability in description logic. As noted before, our benchmark is not a description logic benchmark. We are more concerned about the issue of storing and querying large amount of data that are created for realistic Semantic Web systems. [8] and [16] test the systems with respect to knowledge bases composed of a Tbox and an Abox, which can essentially be viewed as the counterparts of the ontology and the data set in our benchmark respectively. [16] uses both synthetic and realistic Tboxes and uses a synthetic Abox. But its Abox is predetermined and not scalable. In contrast, our benchmark data are generated randomly and can scale to arbitrary size. [8] generates the Abox randomly. However, unlike our benchmark data, the Abox is not customizable and repeatable. It generates the Tbox randomly while our benchmark is based on a realistic ontology.

[3] Our initial investigation suggested that this was due to the difference in the structures of class Person's hierarchies in the two repositories.

Lastly, there exist some query languages for RDF or DAML+OIL such as DQL [9], RQL [17] and TRIPLE [21]. Since none of these has proven dominant, we decided to use a query language that could be easily translated into any of them.

5 Conclusions

In this paper we presented the Lehigh University benchmark. The benchmark is intended to evaluate the performance of DAML+OIL repositories with respect to extensional queries over a large data set that commits to a single realistic ontology. It supports the evaluation of arbitrary repositories. It facilitates the evaluation with synthetically generated data based on the Univ-Bench ontology and allows tests to be conducted on data of arbitrary size. The test queries and performance metrics are designed so that the benchmark could evaluate not only the underlying system and schema but functionalities integrated in the repository such as support for DAML+OIL entailment. Moreover, the benchmark includes a metric for the degree to which a repository returns complete query answers. Consequently, the benchmark could help the user to evaluate different repositories depending on the preference in the tradeoff between logical completeness and system performance. Those interested in obtaining the benchmark and supporting tools should visit http://www.cse.lehigh.edu/~heflin/research/.

Using the benchmark, we have conducted an experiment to evaluate DLDB, a DAML+OIL repository that extends a relational DBMS with description logic inference capabilities. We compared DLDB against two limited versions of it. The experiment showed that the storage mechanism adopted in DLDB plus the inference capabilities added to it could help increase the degree of completeness of query answers. It also showed that there was a tradeoff between the logical completeness and system performance. However the loss in the later appeared to be insignificant in light of the gain in the former. We also expect that the results in this experiment could be used as a baseline for other work on similar systems and schemata.

We intend to explore a number of ideas as future work. First, although designed for DAML+OIL, the benchmark can be easily adapted to OWL [7]. Second, we will use the Lehigh University benchmark to evaluate a variety of Semantic Web repositories with respect to performance and answer completeness. Third, as mentioned, we will examine the scalability of the repositories with even larger data sets. Fourth, we intend to develop another benchmark that includes multiple ontologies and supports ontology integration. Finally, we want to create benchmarks that are based on ontologies with different characteristics.

References

1. R. Agrawal, A. Somani, and Y. Xu. Storage and Querying of E-Commerce Data. In Proc. of VLDB 2001, the 27th International Conference on Very Large Data Bases. 2001.
2. S. Alexaki et al. On Storing Voluminous RDF Description: The case of Web Portal Catalogs. In Proc. of the 4th International Workshop on the Web and Databases (WebDB2001) in conjunction with ACM SIGMOD'01 Conference. 2001.

3. D. Bitton, D. DeWitt, and C. Turbyfill. Benchmarking Database Systems, a Systematic Approach. In Proc. of the 9th International Conference on Very Large Data Bases. 1983.
4. D. Bitton and C. Turbyfill. A Retrospective on the Wisconsin Benchmark. In Readings in Database Systems, Second Edition. 1994.
5. R.G.G.Cattell. An Engineering Database Benchmark. In Readings in Database Systems, Second Edition. 1994.
6. Dan Connolly et al. DAML+OIL (March 2001) Reference Description. http://www.w3.org/TR/daml+oil-reference
7. Mike Dean et al. Web Ontology Language (OWL) Reference Version 1.0. W3C Working Draft 12 November 2002. http://www.w3.org/TR/2002/WD-owl-ref-20021112
8. Quentin Elhaik, M-C Rousset, and Bernard Ycart. Generating Random Benchmarks for Description Logics. In Proceedings of DL'98. 1998.
9. Richard Fikes et al. DAML Query Language (April 2003). http://www.daml.org/2003/04/dql
10. Gensereth M., Fikes R. Knowledge Interchange Format. Stanford Logic Report Logic-92-1, Stanford Univ. http://logic.stanford.edu/kif/kif.html
11. M. Gertz and K. Sattler. A Model and Architecture for Conceptualized Data Annotations. Technical Report, Dept. of Computer Science, University of California, Davis. 2001.
12. Jeff Heflin. Univ1.0 Ontology. http://www.cs.umd.edu/projects/plus/DAML/onts/univ1.0.daml
13. Jeff Heflin and Zhengxiang Pan. Univ-Bench Ontology. http://www.lehigh.edu/~zhp2/univ-bench.daml
14. I. Horrocks. Benchmark Analysis with FaCT. In Proc. of TABLEAUX 2000. 2000.
15. I. Horrocks. The FaCT System. In Automated Reasoning with Analytic Tableaux and Related Methods: International Conference (Tableaux'98). 1998.
16. Horrocks, I. and Patel-Schneider, P. DL systems comparison. In Proceedings of DL'98. 1998.
17. G. Karvounarakis, V. Christophides, D. Plexousakis, and S. Alexaki. Querying Community Web Portals. Technical report, Institute of Computer Science, FORTH, Heraklion, Greece. 2000. http://www.ics.forth.gr/proj/isst/RDF/RQL/rql.pdf
18. A. Magkanaraki et al. Benchmarking RDF schemas for the Semantic Web. In Proc. of First International Semantic Web Conference. 2002.
19. S. Melnik. Storing RDF in a relational database. http://www-db.stanford.edu/~melnik/rdf/db.html
20. Zhengxiang Pan and Jeff Heflin. DLDB: Extending Relational Databases to Support Semantic Web Queries. Submitted to Workshop on Practical and Scaleable Semantic Systems, International Semantic Web Conference 2003.
21. Michael Sintek and Stefan Decker. TRIPLE – A Query, Inference, and Transformation Language for the Semantic Web. In Proc. of First International Semantic Web Conference. 2002.
22. M. Stonebraker et al. The SEQUIOA 2000 Storage Benchmark. In Readings in Database Systems, Second Edition. 1994.
23. World Wide Web Consortium. Resource Description Framework (RDF). http://www.w3.org/RDF/