Performance Analysis of a Deductive Database with a Semantic Web Reasoning Engine: ConceptBase and Racer

Simone A. Ludwig, Craig Thompson, Kristofor Amundson
Department of Computer Science, University of Saskatchewan, Canada
ludwig@cs.usask.ca

Abstract

Knowledge engineering is a discipline concerned with constructing and maintaining knowledge bases to store knowledge of various domains and using the knowledge by automated reasoning techniques to solve problems in domains that ordinarily require human logical reasoning. Therefore, the two key issues in knowledge engineering are how to construct and maintain knowledge bases, and how to reason out new knowledge from known knowledge effectively and efficiently. The objective of this paper is the evaluation of a Deductive Database system with a Semantic Web reasoning engine. For each system a knowledge base is implemented in such a way that comparable performance measurements can be performed. The performance and scalability are evaluated for class and instance queries.

1. Introduction

Knowledge engineering is a discipline concerned with constructing and maintaining knowledge bases to store knowledge of the real world in various domains and using the knowledge by automated reasoning techniques to solve problems in domains that ordinarily require human logical reasoning. Therefore, the two key issues in knowledge engineering are how to construct and maintain knowledge bases, and how to reason out new knowledge from known knowledge effectively and efficiently.

Knowledge-based systems (KBS) use human knowledge to solve problems which normally requires human intelligence. A KBS shell is a software environment containing a knowledge acquisition system, the knowledge base itself, inference engine, explanation subsystem and user interface. The core components are the knowledge base (human knowledge represented by e.g. IF-THEN rules) and the inference engine (forward or backward chaining).

MYCIN [1] is an example of a rule-based expert system which was designed for the diagnosis of infectious blood diseases. MYCIN has been developed without using a modeling framework, opposed to a few frameworks which were developed to help during the knowledge engineering process such as CLIPS (C Language Integrated Production System) [2] or JESS (Java Expert Systems Shell) [3]. CLIPS is a productive development and delivery expert system tool which provides a complete environment for the construction of rule and/or object based expert systems. JESS is a rule engine and scripting environment written in Java. With JESS, one can build software that has the capacity to "reason" using knowledge supplied in the form of declarative rules. JESS uses an enhanced version of the Rete algorithm [4] to process rules which is a very efficient mechanism for solving the difficult many-to-many matching problem. CommonKADS [5] is known for having a structure of the Expertise Model and Model-based and Incremental Knowledge Engineering (MIKE) [6], which relies on formal and executable specification of the expertise model as the result of the knowledge acquisition phase.

Another approach for reasoning is Deductive Databases, where data is described by logical formulas, usually in a restricted subset of first-order logic. These formulas are intended to specify part of the external world relevant to the application at hand, called the application world. Thus, a Deductive Database is a logical representation of the application world. Therefore, the semantics of Deductive Databases are based on mathematical logic. A user queries a Deductive Database by submitting a goal. Goals are also logical formulas. A correct answer to a goal provides values for the variables of the goal that make this query logically follow from the database. Hence, the semantics of query answering in Deductive Databases is based on the notion of logical consequences developed in mathematical logic. Besides formulas specifying the database and queries, a Deductive Database can also contain integrity constraints: logical conditions which the database must satisfy at any given moment [7].

The latest reasoning technology for the Web is the Semantic Web, which vision is to make the Web machine-readable, allowing computers to integrate information and services from diverse sources to achieve the goals of end users. It allows to reason about the content when Web pages and services are augmented with descriptions of their content. Semantic Web technologies are used in many ways to transform the functionality of the Web by enriching metadata for Web content to improve search and management; enriching descriptions of Web services to improve discovery and composition; providing common access wrappers for information systems to make integration of heterogeneous systems easier; and exchanging semantically rich information between software agents. Ontology languages [8] were created to augment data with metadata. The most recent ontology for the Web is called OWL (Web Ontology Language). OWL builds on a rich technical tradition of both formal research and practical implementation.

This research was motivated by the fact that reasoning on the Web becomes ever more important due to the advancement of Web services and service computing on the whole. However, not much research has been conducted into the evaluation of the performance and scalability of reasoning on the Web. Furthermore, no comparison between an established reasoning tool, namely the deductive database, has been done. The objective of this paper is the evaluation of a Deductive Database system with a Semantic Web reasoning engine. For each system a knowledge base is implemented in such a way that comparable performance measurements can be performed.

The paper is outlined as follows. In Section 2, both systems, ConceptBase and Racer are described. In Section 3, the knowledge base, queries, measurement methodology and setup are outlined. Section 4 presents the performance analysis of both systems exploring the load time and the scalability of classes and instances. The findings and conclusions are given in Section 5.

2. Description of Both Systems

ConceptBase was chosen as the Deductive Database system to compare with the Semantic Web reasoning engine Racer. The two systems are described in more details in the subsections below.

2.1. ConceptBase

ConceptBase has been used in a number of applications at various universities in Europe. The ConceptBase system, developed since 1987, seeks to combine deductive rules with a semantic data model based on Telos [9] (described further below). The system also provides support for integrity constraints [10]. ConceptBase is free software available for download, and the user interface is java based. Furthermore, ConceptBase uses the client-server architecture, and has a fairly extensive Application Programming Interface (API) for writing clients in Java, C or C++.

ConceptBase is a deductive object-oriented database management program intended for conceptual modeling. It uses O-Telos which is a version of the logical knowledge representation language Telos, which includes deductive and object-oriented features. O-Telos is based on Datalog, which is a subset of Prolog.

ConceptBase allows for logical, graphical and frame views of databases. The ConceptBase graph editor allows one to visualize the relationships in the database, as well as adding and modifying the classes, individuals, and relationships. Queries are represented as classes that have membership constraints. Within the database, all classes, instances, attributes, rules and constraints are represented as objects that may be updated at any time. However, there is not an option to cascade changes, so it is easy to add information at any time, but it can be difficult to remove information.

2.2. Semantic Web Technologies: Protégé and Racer

The Semantic Web technology used to create an ontology to represent the application domain was Protégé [11], a Java-based, free ontology editor developed by Stanford Medical Informatics at the Stanford University School of Medicine. It provides a knowledge base that allows the user to create formal rules for a knowledge representation system to reason through. After developing a taxonomy and creating rules the ontology can be exported in OWL format, which is similar to XML in syntax and includes the descriptions of the classes and individuals along with their explicit relationships. Protégé also provides a Java API that allows OWL files to be imported and represented as Java classes. The API has the capability to connect to a knowledge representation system, such as RACER (Renamed ABox and Concept Expression Reasoner) [12], allowing implicit relationships to be found.

RACER is commercial software developed by RACER Systems and was used for this research investigation. This software is capable of reasoning through Description Logic TBoxes (subsumption, satisfiability, classification) and ABoxes (retrieval, tree-conjunctive query answering using an XQuerylike syntax), such as the ones that are created using Protégé and exported in the OWL format.

3. Evaluation

In order to perform a comparison analysis of Racer and ConceptBase, a knowledge base was implemented in both systems. Queries were chosen which return the same results to evaluate class and instance queries. The measurement methodology and setup are described below.

3.1. Knowledge Base

The knowledge base / ontology used for the evaluation is an extension of the pizza ontology supplied with Protégé.

Table 1. Ontology description of scaling classes

Ontology size	Number of classes	File size Racer (in KB)	File size ConceptBase (in KB)
1	263	279	27
2	495	565	54
3	727	869	82
4	959	1189	109
5	1191	1536	137
6	1423	1897	163
7	1655	2280	191
8	1887	2683	219
9	2119	3105	246
10	2351	3553	273

The ontology contains classes describing pizzas and ingredients, as well as sandwiches and salads. The dishes (pizzas, sandwiches, salads) were defined in terms of the ingredients they contain. All subclasses in the ontology were given instances, and in some cases higher level classes had instances, so there are nearly as many instances as classes. Some dishes were defined to describe specific foods, such as a BLT (Bacon Lettuce Tomato) sandwich, other dishes such as *vegetarianPizza* were defined to be any pizza without meat or fish. The classes describing specific foods were given necessary conditions, for example, this pizza must have mozzarella as a topping. The other classes, such as *vegetarianPizza*, were given necessary and sufficient conditions, meaning that any pizza that had no meat or fish would be considered a *vegetarianPizza*. Thus, the classes that met the necessary and sufficient conditions would be subsumed, creating an inferred hierarchy of classes.

ConceptBase on the other hand, required a slightly different modeling technique. It is not possible to create an inferred class hierarchy, thus, in order to have similar reasoning capabilities to the Protégé ontology, *queryClasses* were used. Query classes have constraints describing which individuals may be members of the query class. Thus, with the vegetarian pizza example, members of the vegetarian pizza query class were defined to be any individual that did not have meat, or fish, as an ingredient.

Table 2. Ontology description of scaling instances (number of classes fixed to 263)

Ontology size	Number of instances	File size Racer (in KB)	File size ConceptBase (in KB)
1	217	305	46
2	434	321	65
3	651	348	84
4	868	375	104
5	1085	402	123
6	1302	433	142
7	1519	454	162
8	1736	480	181
9	1953	507	200
10	2170	542	220

As knowledge bases consist of classes and instances, the investigation will only focus on class and instance reasoning. In order to measure how good both systems scale, we expanded the ontologies in two directions; (1) scaling of classes and (2) scaling of instances. Table 1 and 2 show the properties of the different ontology sizes used for this investigation. Table 1 contains 10 different ontology sizes, whereby the number of classes is increasing with the size without containing any instances. For the scaling of instances, the class structure of the size 1 ontology (Table 1) is fixed to 263 classes for all different instance ontology sizes.

3.2. API and Queries

The ConceptBase API provides methods to ‘tell’ files to the ConceptBase server, retrieve a named class or individual, find instances of a class or query a class, retrieve attributes of classes or individuals, retrieve superclasses and subclasses, find the class that an individual belongs to, and get generalizations and specializations from a class. Additionally, several Boolean operations are provided for testing the relationships between classes, or instances, such as *isSuperclassOf* or *isExplicitInstanceOf*. There appeared to be many methods that returned the same, or very similar results in different formats, such as newline delimited, or comma delimited. The redundant queries in ConceptBase returned the same classes, but in different formats, e.g., subclasses can be returned in ConceptBase code syntax, or as a string with one class per line, or with all classes on one line separated by commas, or as a hashset, depending on what the user want to do with the subclasses. Attributes in ConceptBase are tied directly to the class they represent, so all information about attributes is gained through the appropriate class, or instance. The useful methods for obtaining information from the database can be found in the *ICBclient* and *ITelosObjectSet* classes.

The Protégé API allows the user to find descendant classes, classify the taxonomy, compute the inferred hierarchy, compute the inferred types of all individuals, retrieve ancestor classes, retrieve equivalent classes, retrieve subclasses, find individuals belonging to a class, determine the subsumption relationship between two classes, return the superclass of a class, get sub properties, get inverse properties, return the inferred equivalent classes, get the inferred subclasses, get the inferred superclasses, maximum and minimum cardinalities of properties, determine if subclasses are disjoint, determine if a class has a superclass, return the name of an instance, return the namespace of the ontology, return a list of the possible rdf properties, and return rdf types. Properties in Protégé are independent of classes and instances, and thus may be queried directly. The useful methods for gaining information about the model were spread across several classes in the API, namely, *ProtegeOWLReasoner*, *RDFProperty*, *OWLProperty*, *OWLNamedClass* and *OWLIndividual*. Among these classes, there seemed to be several redundant methods. This is because *OWLProperty* inherits from *RDFProperty* and therefore has all the same methods, plus a few more. *ProtegeOWLReasoner* and *OWLNamedClass* have some methods with the same results, the difference is that *ProtegeOWLReasoner* calls RACER, whereas *OWLNamedClass* uses the results from the last time the reasoner was used.

The main type of reasoning of a knowledge base can be divided into two categories, class and instance reasoning. In order to perform a fair analysis of these systems, equivalent queries existent in both systems which perform the same type of reasoning were chosen: Query 1 and 2 are class queries, and query 3 and 4 are instance queries.

Query 1 returns all subclasses belonging to a particular class: *getDescendentClasses* (Racer); *getAllSubclassesOf* (ConceptBase).

Query 2 returns the superclasses of a particular class: *getSuperClasses* (Racer); *getExplicitSuperClasses* (ConceptBase).

Query 3 returns all individuals that are members of a particular class: *getIndividualsBelongingToClass* (Racer); *getAllInstancesOf* (ConceptBase).

Query 4 returns all classes that an individual or an instance belongs to: *getIndividualTypes* (Racer); *getClassificationOf* (ConceptBase).

3.3. Methodology

Bash scripts were used to automate all the measurement runs. The process for each measurement was as follows: start Racer or the ConceptBase server, run the java query, and close Racer or ConceptBase server to clear the cache. This process was repeated 30 times (to guarantee normal distribution) for each query. The Java query file used to perform a query would start by loading the data model into Racer or the ConceptBase server. Then, the java method *System.nanoTime* was used immediately before and after the query, and the difference was calculated to estimate the performance of the query. Each time the java program was executed it would perform only one query, in order to avoid caching issues across queries. *System.nanoTime* was found to give results with a higher precision than *System.currentTimeMillis*, especially as several of the queries took less than one millisecond to execute.

3.4. Measurement Setup

The following measurement setup was used for this investigation:

- Hardware configuration (Lenovo M55 with 2.4GHz Intel Core2 CPUs and 2GB of RAM; no hyperthreading).
- Software configuration (Mandriva Linux 2008.1; Java 1.6.0_03; latest versions of ConceptBase 7.1, Protégé 3.4 and Racer 1.9.2.)

4. Results

The evaluation was performed as follows. First, the load times for loading the different ontologies into memory are measured. Afterwards, the scalability of classes and instances are evaluated.

4.1. Load Time of Different Ontology Sizes

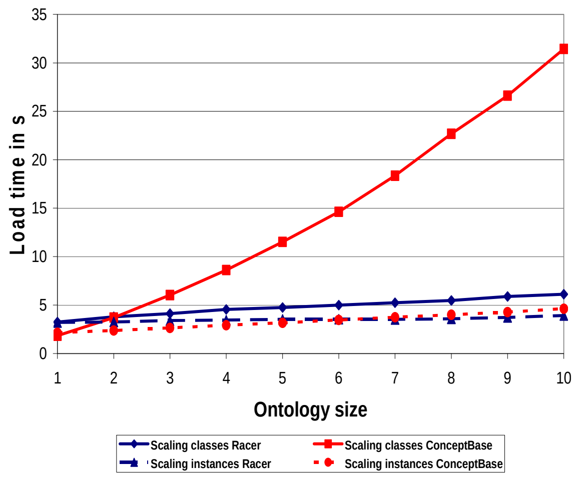

Figure 1. Load time of Racer for scaling of classes and instances

Before queries are run in both Racer and ConceptBase, the knowledge base or ontology needs to be loaded into memory first. Figure 1 shows the load time in seconds for increasing ontology sizes. Two distinctions are made here for either scaling of classes or instances. The scaling of Racer shows a linear distribution with increasing ontology sizes, whereby the scaling of classes has a greater impact on the performance than the scaling of instances. The scaling of classes has a gradient of 0.3, whereas the scaling of instances has a gradient of 0.07. The load time for ConceptBase has a slightly different distribution. The scaling of instances seems to be linear; however, the scaling of classes follows a quadratic distribution. The query times for the scaling of classes are also larger than for the scaling of instances as also observed for Racer. Comparing both systems it can be concluded that the load time of ConceptBase is greater by a factor of 3.01 for the scaling of classes, but is almost similar for the scaling of instances with a factor of 0.95.

4.2. Scaling of Classes

Looking at how the performance scales with increasing ontology sizes (the instance queries would not make sense when querying an ontology without instances), Figure 2 shows the queries run in Racer and ConceptBase (*getSuperClasses*, *getAllSubclassesOf* and *getSuperclassesOf* all have similar query times). It is observed, that both queries, *getDecendentClasses* and *getSuperclasses*, scale in a similar fashion with a quadratic distribution. This is because the same operation is performed, that is the classification of a new concept. Racer computes more than is required in order to answer these particular class queries. ConceptBase on the other hand shows the measurements of the similar queries with a linear distribution. Instead of both queries scaling in a simiar fashion as in Racer, the query time for subclasses is higher than for superclasses. It appears that the performance is dependent on the number of return values. *getAllSubclassesOf* returns 31 to 238 subclasses for ontology size 1 and 10 respectively, while *getSuperclassesOf* returns only 1 superclass for all ontology sizes.

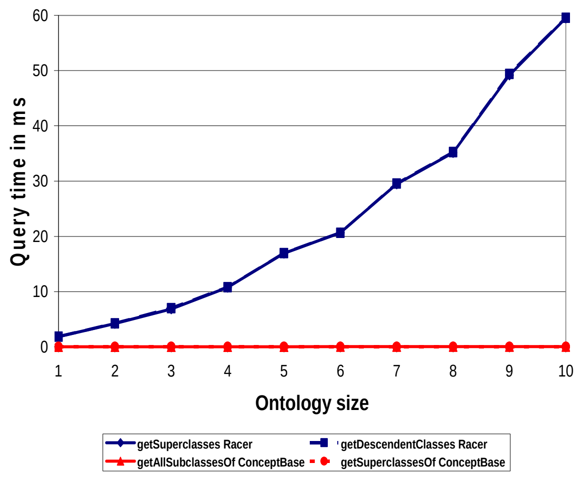

Figure 2. Query time of Racer for scaling of classes

Comparing the class queries executed in Racer and ConceptBase, it shows that ConceptBase scales much better than Racer.

4.3. Scaling of Instances

Figure 3 shows the linear distribution of query times for scaling of instances. For Racer, it shows that the query times for *getIndividualBelongingToClass* are higher than for *getIndividualTypes* queries. *getIndividualTypes* looks at a specific individual and returns all classes of which it is an instance of, whereas *getIndividualsBelongingToClass* has to consider all individuals. The implementation of the operations seem to be quite different and therefore the result can be seen in the query times of both queries in Racer. The instance query, *getAllInstancesOf*, performed in ConceptBase shows a quadratic distribution, whereas the *getClassificationOf* query shows a linear gradient. *getAllInstancesOf* takes longer as the return values range between 47 to 256 for ontology size 1 to 10 respectively, whereas *getClassificationOf* returns always only one return value. The direct comparison of the query times regarding the scaling of instances of both systems shows that ConceptBase performs better than Racer.

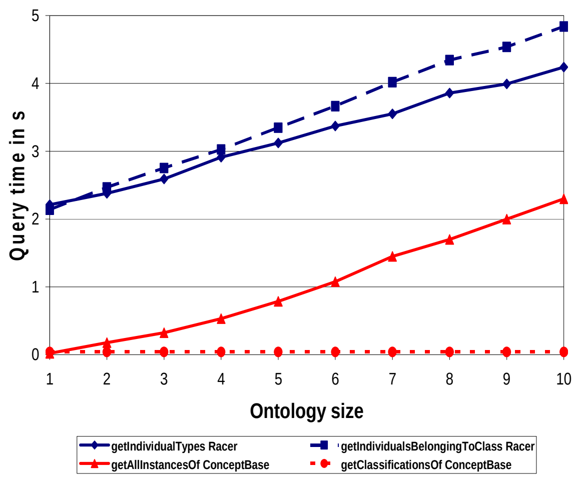

Figure 3. Query time of Racer for scaling of instances

5. Conclusion

This paper evaluated a Deductive Database system and a Semantic Web reasoning engine - ConceptBase and Racer. For each system a knowledge base was implemented in such a way that comparable performance measurements could be performed.

The findings revealed that the reasoning capabilities in Racer are richer. Furthermore, queries in ConceptBase run much faster than in Racer. The factors were 61 and 7 for ontology sizes 1 and 10 respectively for ontologies with instances. On the other hand however, the load time to load the different ontologies was better in Racer by a factor of 3.01 for the scaling of classes, but was almost similar for the scaling of instances for which the factor measured was 0.95. The load time for Racer is linear, whereas the load time for ConceptBase seems to have a quadratic growth function. The scaling of classes revealed that class queries take much longer in Racer than in ConceptBases as in Racer all consistencies are being checked before a class query is being performed, measured by a factor of 470. The growth function for Racer for the class queries is quadratic, whereas the growth function for ConceptBase is linear. The scaling of instance queries showed a better performance for ConceptBase than for Racer by a factor of 4.8. However, it seems that ConceptBase is more affected by string processing of the return values of the queries. This means that if a large amount of result values are returned, the performance decreases in ConceptBase, whereas Racer is not affected by this.

Considering that ontologies are developed incrementally, adding a relatively small increment to a large ontology has a great effect for the loading of this ontology into memory for ConceptBase, whereas the class and instance queries in Racer will have a greater performance reduction than ConceptBase.

As reasoning on the Web has seen a steady increase in the past several years, this evaluation shows that Web reasoning has to speed and scale up with technologies existing for many decades such as deductive databases.

6. References

[1] Shortliffe, E. H., “MYCIN: Computer-Based Medical Consultations”, Elsevier Press, New York, 1976.

[2] CLIPS Website, Last retrieved April 2009 from http://www.ghg.net/clips/CLIPS.html.

[3] JESS Website, Last retrieved April 2009 from http://herzberg.ca.sandia.gov/jess/.

[4] Forgy, C. L., “Rete: A Fast Algorithm for the Many Pattern/Many Object Pattern Match Problem”. Artificial Intelligence, 19(1982) 17-37.

[5] Schreiber, A. T., Wielinga, B.J., de Hoog, R., Akkermans, H., and van de Velde, W., “CommonKADS: A Comprehensive Methodology for KBS Development”, IEEE Expert, December 1994, 28-37.

[6] Angele, J., Fensel, D., and Studer, R., “Developing Knowledge-Based Systems with MIKE”, Journal of Automated Software Engineering, Volume 5, Number 4, pp. 389-418(30), 1998.

[7] Voronkov, A., Chapter 1, Lecture notes on Deductive Databases, 2002. http://www.voronkov.com/dresden/2002/chapter_1.ps.

[8] Gruber, T. R., “Toward principles for the design of ontologies used for knowledge sharing”. Journal of Human-Computer Studies, Volume 43, Issue 5-6 Nov./Dec. 1995, Pages: 907-928, 1993.

[9] Jeusfeld, M. and Jarke, M., “From relational to object-oriented integrity simplification”, Proc. Of Deductive and Object-Oriented Databases 91. Springer-Verlag, 1991.

[10] Jeusfeld, M. and Staudt, M., “Query optimization in deductive object bases”. In G. Vossen J.C. Freytag and D. Maier, editors, Query Processing for Advanced Database Applications. Morgan-Kaufmann, 1993.

[11] Protégé Website, Last retrieved April 2009 from http://protégé.stanford.edu.

[12] Racer Website, Last retrieved April 2009 from http://www.racer-systems.com.