Knowledge-Based Systems xxx (2010) xxx–xxx

Contents lists available at ScienceDirect

Knowledge-Based Systems

journal homepage: www.elsevier.com/locate/knosys

Short communication

Comparison of a Deductive Database with a Semantic Web reasoning engine

Simone A. Ludwig

Department of Computer Science, University of Saskatchewan, Canada

ARTICLE INFO

Article history:
Received 11 November 2008
Received in revised form 5 November 2009
Accepted 5 April 2010
Available online xxxx

Keywords:
Knowledge engineering
Semantic Web
Performance comparison
ConceptBase
Racer
Protege

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 comparison and evaluation of a Deductive Database system (ConceptBase) with a Semantic Web reasoning engine (Racer). For each system a knowledge base is implemented in such a way that a fair comparison can be achieved. Issues such as documentation, feasibility, expressiveness, complexity, distribution, performance and scalability are investigated in order to explore the advantages and shortcomings of each system.

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.

Automated reasoning is concerned with the building of computing systems that automate this process. Although the overall goal is to automate different forms of reasoning, the term has largely been identified with valid deductive reasoning as conducted in mathematics and formal logic. In this respect, automated reasoning can be seen as mechanical theorem proving. Building an automated reasoning program means providing an algorithmic description to a formal calculus so that it can be implemented on a computer to prove theorems of the calculus in an efficient manner. Important aspects of this exercise involve defining the classes of problems the program will be required to solve, deciding what language will be used by the program to represent the given information as well as new information inferred by the program, and specifying the mechanism that the program will use to conduct deductive inferences.

Knowledge-based systems (KBS) use human knowledge to solve problems which normally requires human intelligence. These systems have been used in industry, finance and government for many years. Approaches have varied from simple rule-based systems to more complex models using fuzzy logic and artificial neural networks and incorporating probability theory, pattern recognition and multi-variate analysis techniques. The 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. It provides a doctor with therapeutic advice in a convenient, user-friendly manner. MYCIN has a number of characteristics common to expert systems, including MYCIN can perform at a level equivalent to human experts in the field and considerably better than junior doctors; MYCIN's knowledge consists of about 450 independent rules of IF-THEN form derived from human knowledge in a narrow domain through extensive interviewing of experts; the knowledge incorporated in the form of rules is clearly separated from the reasoning mechanism. 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 is small, light, and one of the fastest rule engines available. JESS uses an enhanced version of the

E-mail address: ludwig@cs.usask.ca

0950-7051/$ - see front matter © 2010 Elsevier B.V. All rights reserved.
doi:10.1016/j.knosys.2010.04.005

Please cite this article in press as: S.A. Ludwig, Comparison of a Deductive Database with a Semantic Web reasoning engine, Knowl. Based Syst. (2010), doi:10.1016/j.knosys.2010.04.005

Rete algorithm [4] to process rules and is a very efficient mechanism for solving the difficult many-to-many matching problem.

KBS have been developed as modeling frameworks for the knowledge engineering approach. CommonKADS [5] is known for having a structure of the Expertise Model and Model-based and Incremental Knowledge Engineering (MIKE) [6], and relies on formal and executable specification of the expertise model as the result of the knowledge acquisition phase. CommonKADS supports most aspects of a KBS development project, such as project management organizational analysis (including problem/opportunity identification), knowledge acquisition (including initial project scoping), knowledge analysis and modeling, capture of user requirements, analysis of system integration issues, and knowledge system design. MIKE integrates semiformal and formal specification techniques together with prototyping into a coherent framework. All activities in the building process of a knowledge-based system are embedded in a cyclic process model.

Two different approaches concerned with automated reasoning, namely Deductive Databases and the state-of-the-art Semantic Web technology are being explored in this paper. An introduction of the two systems follows.

The paradigm of Deductive Databases evolved during the 1980s, based on the ideas developed in relational databases and logic programming. Deductive Databases have been developed with the aim of increasing the expressive power of relational query languages. The technology of Deductive Databases extends the relational database technology based on the ideas developed in logic programming. Thus, Deductive Databases inherit the ideas and properties of both their predecessors: relational databases and logic programming.

In Deductive Databases, 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. The semantics of integrity constraints is also based on mathematical logic [7].

The Semantic Web 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 becomes possible to reason about the content when Web pages and services are augmented with descriptions of their content. The potential impact is huge, representing a reinvention of the world's computing infrastructure on at least the scale of the original Web. Semantic Web technology could be 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. The technical basis for much of OWL is the part of the formal knowledge representations field known as Description Logic (DL). DL is the main formal underpinning of such diverse kinds of knowledge representation formalisms as semantic nets, frame-based systems, and others.

The Semantic Web envisions the idea of having Web resources augmented with semantics, so that machines, e.g. via software agents, can interact with each other in order to solve a common goal. The process has to be fully automated, integration of heterogeneous systems has to be made much easier, and reuse of data across various applications has to be made possible to ensure the successful delivery of the vision. The aim for the prosperity of the Semantic Web is to develop open standards and tools in order to make it as successful as the Web. These standards, and the tools developed to embody them, must be easy to use and fault tolerant to allow everyone to contribute and to bring the Semantic Web vision to reality.

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 a comparable evaluation can be achieved. Issues such as documentation, feasibility, expressiveness, complexity, distribution, performance and scalability are investigated in order to explore the advantages and shortcomings of each system.

The paper is outlined as follows. Section 2 describes both systems, ConceptBase and Racer. In Section 3, the comparison criteria are explored and summarized. Furthermore, this section also presents the performance analysis of both systems exploring the direct comparison of queries, load time, and scalability of classes and instances. The findings and conclusions are given in Section 4.

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 following subsections.

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, thus it is easy to add information at any time, but it can be difficult to remove information.

2.2. Semantic Web technology: Protégé and Racer

The Semantic Web technology used to create an ontology to represent the application domain was Protégé [11], a Java-based,

Please cite this article in press as: S.A. Ludwig, Comparison of a Deductive Database with a Semantic Web reasoning engine, Knowl. Based Syst. (2010), doi:10.1016/j.knosys.2010.04.005

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 the Web Ontology Language (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 a 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 a XQuery-like syntax), such as the ones that are created using Protégé and exported in the OWL format.

2.3. Comparison of different logics

Both logics, O-Telos which is the logic used in ConceptBase and OWL used in Racer, are based to some extent on RDF (Resource Description Framework). RDF(S) is an extension of RDF and represents a simple yet powerful model of semantic networks. In RDF(S) every statement is expressed as a binary predicate on ground arguments, with nodes in the graph denoting arbitrary resources and literals which are used as subjects and objects of statements, whereas arcs denote specific relationships between two of these nodes [13].

O-Telos is derived from the knowledge representation language Telos. While Telos was geared more to its roots in artificial intelligence, O-Telos is more geared to database theory, in particular deductive databases. O-Telos is an object-oriented meta-modeling language that provides facilities for unrestricted meta modeling levels. ConceptBase implements a powerful query and reasoning (rules and constraints) mechanism based on Datalog.

OWL provides reasoning capabilities for the Web by supporting XML, RDF and RDF(S) and providing additional vocabulary along with a formal semantics. OWL has three increasingly-expressive sublanguages: OWL Lite, OWL DL and OWL Full. OWL DL (Description Logics) supports users who want the maximum expressiveness while retaining computational completeness and decidability. OWL DL is used for the evaluation.

The main difference between O-Telos and OWL, is that O-Telos is based on quadruples where the additional components identifies the statement, while OWL has to use special link types to reify triple statements, i.e. to make statements about statements, O-Telos statements are simply referred to by their identifier. Furthermore, O-Telos uses Datalog, including negation and recursion as the underlying logic. For this logic, certain problems are undecidable, e.g. whether two query classes are disjoint, equivalent, or one is a subset of the other, cannot be decided in general for Datalog. Therefore, queries existent in OWL such as *isDisjointTo* are not available in O-Telos due to the usage of Datalog.

3. Comparison

The ontology used for the evaluation is an extension of the pizza ontology supplied with Protégé [14]. 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, thus 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. Fig. 1 shows a screenshot of part of the food ontology created using the Protégé editor.

ConceptBase on the other hand, required a slightly different modeling technique. As it is not possible to create an inferred class hierarchy, *queryClasses* were used in order to have similar reasoning capabilities as Racer. 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.

As knowledge bases consist of classes and instances, the investigation focuses on class and instance reasoning. In order to measure how good both systems scale, we have expanded the ontologies in two directions; (1) scaling of classes and (2) scaling of instances. Tables 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 sizes.

3.1. Documentation

The documentation accompanying ConceptBase consists of a tutorial, and a user manual. In addition, there is both a FAQ and a forum online, and an API, with its own documentation available upon request. The ConceptBase tutorial is quite short, and there is a large leap between the tutorial and the other provided documentation. The tutorial gives the reader a good start with ConceptBase, but often leaves tasks up to the reader to solve. The tutorial does not have enough depth, whereas the user manual has ample depth but only few examples are given. The FAQ is useful, providing answers to common questions about getting ConceptBase running properly. The forum contains a How-To section, a related documents section, and section for third party software extensions. The javadocs are useful, however, they only describe what the parameters and return values for methods are, and give little information about what the methods mean, or how they work. In addition to the javadocs, there is similar documentation describing the C and C++ APIs.

The documentation accompanying Protégé is expansive and the community is quite active. For new users the Protégé FAQ provides a quick way to get answers in order to start creating ontologies using the software. The FAQ section provides information about installation, general UI concerns, knowledge representation, database support, plug-ins and licensing. There are also separate FAQ sections for the Protégé-Frames editor and the Protégé-OWL editor. Accompanying the documentation are tutorials that provide an excellent way to learn how to develop ontologies, set up client/server capabilities, collaborative ontology development and graphing ontologies. There are comprehensive courses offered for Protégé and Protégé-OWL as well as consulting. Furthermore, there are four separate mailing lists (protégé-users, protégé-discussion, protégé-owl, and protégé-beta) that offer help from the community members.

Please cite this article in press as: S.A. Ludwig, Comparison of a Deductive Database with a Semantic Web reasoning engine, Knowl. Based Syst. (2010), doi:10.1016/j.knosys.2010.04.005

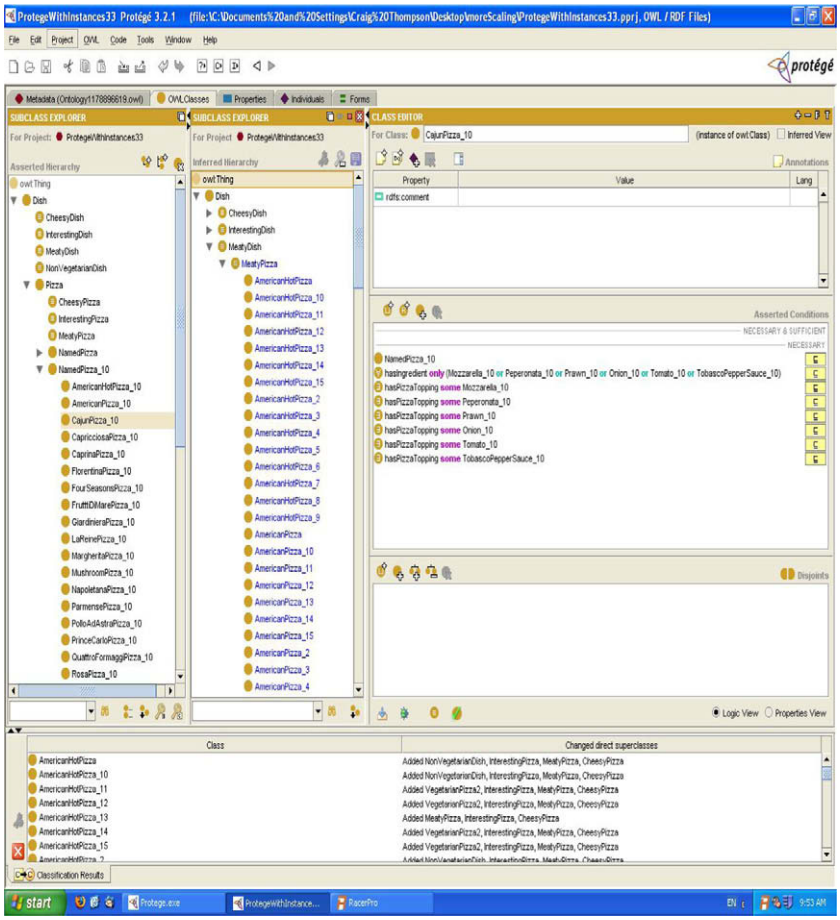

Fig. 1. Food ontology shown in Protégé (same ontology implemented also in ConceptBase).

Table 1
Ontology description of scaling classes.

Ontology size	Number of classes	File size (kB) Racer	File size (kB) ConceptBase
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

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

Ontology size	Number of instances	File size (kB) Racer	File size (kB) ConceptBase
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

3.2. Feasibility and expressiveness

The Protégé application has a query tab which allows the user to write queries that return individuals that match certain constraints, however there is no documentation of this feature, and the user created queries are not stored in the OWL file. Therefore, the user defined queries cannot be accessed from the Protégé Java API.

ConceptBase operates under a closed world assumption. This assumption states that all relevant information is contained within the database; there are no missing pieces of information. Thus, ConceptBase allows a "cheese pizza" (pizza with cheese on it) to be a vegetarian pizza. However, in Protégé, one must specifically state that a pizza has cheese AND ONLY cheese, for it to be a vegetarian pizza. This difference stems from Protégé's open world assumption. This assumption implies that anything is possible, unless specifically denied in the ontology. Therefore, unless specifically limited, a pizza could have any number of toppings and still be a cheese pizza, by virtue of having cheese as a topping. Thus, in Protégé one must state that a cheese pizza has cheese, and only has cheese.

Protégé allows for the modeling of both classes and instances. Additionally, any number of meta-classes may be implemented. An instance is an individual member of some class. Thus, any given class can have subclasses, and instances. ConceptBase on the other hand, allows for creation of subclasses and instances of classes, but also subclasses and instances of instances. The notion of an instance of an instance, or a subclass of an instance is difficult to understand, and makes learning how to create simple databases much more complex.

Please cite this article in press as: S.A. Ludwig, Comparison of a Deductive Database with a Semantic Web reasoning engine, Knowl. Based Syst. (2010), doi:10.1016/j.knosys.2010.04.005

ConceptBase allows for the use and comparison of numerical data, as well as string data types. Protégé on the other hand, does not allow for this. For example, one can express that a person's salary is $50,000 quite easily in ConceptBase, but this is not possible using Protégé. Protégé does not support a numerical data type for the range or domain of properties. Protégé only allows for the domain and range of a property to be classes within the ontology. In ConceptBase, attributes specified within a class do not need to be instantiated. For example, the class "person" could have the attribute "name" which will be a string; one can create an instance of person, and not define a name. Additionally, the name can be added to the instance later. Finally, if a class has an attribute "name" there can be multiple instances of this attribute, thus people can have multiple names, or in the case of our study, dishes can have multiple ingredients.

Protégé properties are defined by their domain and range, as well as whether they are functional, inverse functional, symmetric, or transitive. Properties can then be applied to a class, indicating that members of the class must satisfy the property.

ConceptBase does not allow properties to be defined outside of a class. All properties are attributes of the class they describe.

3.3. Complexity and usability

The code generated by the Protégé GUI is difficult to read, as it is an adaptation of XML. Also, the application was unstable with large ontologies, it would often fail to save correctly, or hang. To ensure that scaling was done correctly, the Protégé program was used rather than using copy and paste to extend the ontology. Thus, creating the larger ontologies for scalability testing was an extremely time consuming process.

The ConceptBase code was easy to read, and the program ran smoothly. There was no generated code to deal with, only the code which was written, so scaling up the databases was quite quick. However, removing concepts from the database was difficult, because there is no 'cascade' option. Thus, if one wanted to remove a base concept from the database one would need to remove all the concepts that referenced the base concept first. Databases using SQL solve this problem with the DROP CASCADE function.

Some queries would time out or take 10 times as long as other queries, it seems that there is very little optimization for some queries. The Protégé OWLViz tab allowed for visualizing the asserted and inferred hierarchy of the ontology, however, there were no options to make changes to the ontology from the OWLViz tab. The ConceptBase graph editor had functionality to view and change the data in the database. Also, the ConceptBase graph editor allows dragging and moving of classes or instances, whereas the Protégé OWLViz tab did not allow the user to sort the classes manually.

ConceptBase is intended as a main memory database, and therefore, the developers warn against using it for databases larger than the computers main memory. Thus, ConceptBase is not well suited as mass storage databases. Also, unlike larger commercial database management systems, ConceptBase has minimal support for recovery, and for multiple user usage.

ConceptBase allows for a fixed number of meta-classes, but currently, an unlimited number of meta-classes are allowed, thus models can be subsumed into larger databases by adding another meta-class level of abstraction, just as with Protégé.

3.4. Distribution

Semantic Web technology such as Protégé and Racer were built with the aim of automating distributed service computing and therefore are distributed by nature. ConceptBase is a client/server installation which means that the knowledge base needs to reside on the server machine. However, to enable distributed knowledge bases, research has been done distributing the data for Deductive Database systems.

Mohania and Sarda [15,16] presented a three level architecture for a distributed Deductive Database system which extends the power of distributed database systems to include deductive capabilities. It allows both knowledge-based and data-based information to be shared by the sites of a computer network in order to process user queries.

Lim and Ng [17] proposed a rule and data allocation algorithm for distributed Deductive Database systems. Rule and data allocation aims to take full advantage of concurrent execution of rules which is an important design issue in distributed Deductive Database systems. However, rule allocation algorithms, which maximize concurrent rule execution with minimal rule and data replication and communication cost in distributed Deductive Database systems, are lacking. The proposed algorithm considers rule and data allocation as a single problem since the two are mutually dependent. The algorithm determines the minimal replication of rules and base relations that should reside at different sites in a distributed database system.

Even though research has been done regarding the distribution of distributed Deductive Database systems, ConceptBase has been implemented as a local installation only.

3.5. Performance and scalability

In order to perform a comparison analysis of ConceptBase and Racer, 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.

3.5.1. APIs 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

Please cite this article in press as: S.A. Ludwig, Comparison of a Deductive Database with a Semantic Web reasoning engine, Knowl. Based Syst. (2010), doi:10.1016/j.knosys.2010.04.005

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. Two class queries (1 and 2) and two instance queries (3 and 4) were selected:

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).

3.5.2. Methodology

Bash scripts were used to automate all 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 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* gives results with a higher precision than *System.currentTimeMillis*, especially as several of the queries took less than 1 ms to execute.

3.5.3. Measurement set-up

The following measurement set-up was used for this investigation.

- Hardware configuration:
- Lenovo M55 with 2.4 GHz Intel Core2 CPUs and 2 GB 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.

3.5.4. Results

The evaluation was performed as follows. First, the queries executed in both systems are compared. Then the load times for loading the different ontologies into memory are measured. Afterwards, the scalability of classes and instances are evaluated.

3.5.4.1. Query comparison. For the direct comparison, the individual- and class-type queries were measured using ontology size 1 with instances and the results are shown in Fig. 2. It shows that all queries take longer in Racer than in ConceptBase.

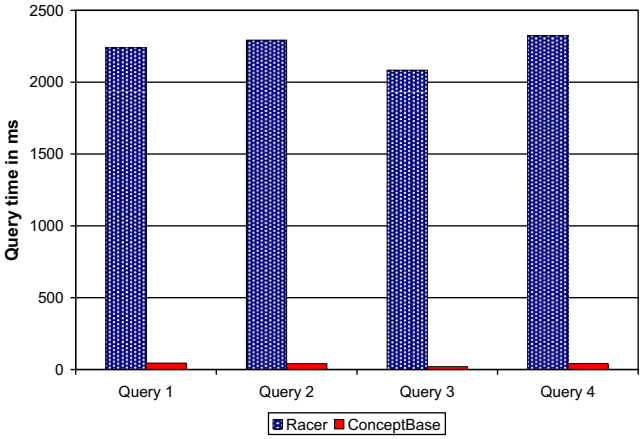

Fig. 2. Comparison of queries for Racer and ConceptBase for ontology size 1 with instances.

For the given ontology size 1 the factor is 61, for ontology size 10 the factor reduces to 7 as the third query, returning instances of a class, is also increasing with increasing ontology sizes. The actual query values for both sizes of ontologies are given in Table 3.

3.5.4.2. Load time of different ontology sizes. Before queries are applied in both Racer and ConceptBase, the knowledge base or ontology needs to be loaded into memory first. Fig. 3 shows the load time in seconds for increasing ontology sizes. Two distinctions are made here for either scaling the classes or the individuals. Both curves show a linear distribution with increasing ontology sizes, however, the scaling of the classes has a greater impact on the per-

Table 3
Query times of four queries in Racer and ConceptBase.

		Query times in ms	
		Racer	ConceptBase
Ontology size 1 with instances	Query 1	2239.63	44.51
	Query 2	2293.48	41.01
	Query 3	2083.87	20.51
	Query 4	2323.39	41.29
Ontology size 10 with instances	Query 1	3848.46	44.82
	Query 2	3841.95	41.03
	Query 3	4835.92	2312.83
	Query 4	4220.36	41.55

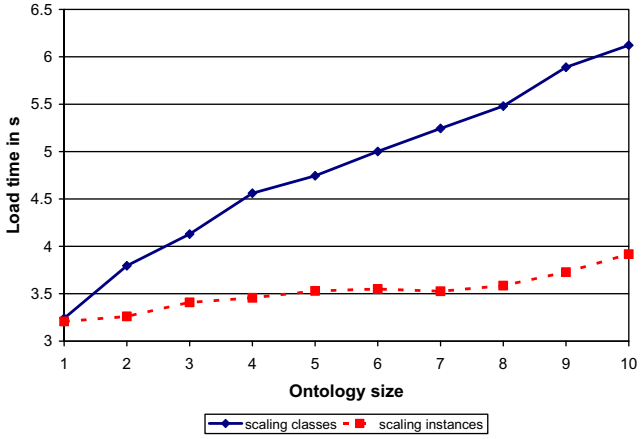

Fig. 3. Load time of Racer for scaling of classes and instances.

Please cite this article in press as: S.A. Ludwig, Comparison of a Deductive Database with a Semantic Web reasoning engine, Knowl. Based Syst. (2010), doi:10.1016/j.knosys.2010.04.005

formance than the scaling of the instances. The scaling of classes has a gradient of 0.3, whereas the scaling of instances has a gradient of 0.07.

Fig. 4 shows the same measurements for ConceptBase with 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 was 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 where the factor is 0.95.

3.5.4.3. Scaling of classes. Looking at how the performance scales with increasing ontology sizes, Fig. 5 shows two queries run in Racer (the instance queries would not make sense when querying an ontology without instances). It is observed, that 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, shown in Fig. 6. Instead of both queries scaling in the same fashion as in Racer, the query time for subclasses is higher than for superclasses. It looks like the performance is dependent on the number of return values. *getAllSubclassesOf* returns 31–238 subclasses for ontology size 1 and 10, respectively, while *getSuperclassesOf* returns only 1 superclass for all ontology sizes.

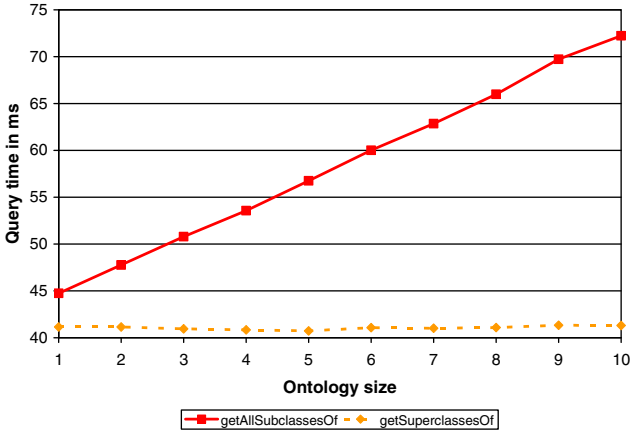

Fig. 6. Query time of ConceptBase for scaling of classes.

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

3.5.4.4. Scaling of instances. Fig. 7 shows the linear distribution of query times for scaling of instances. It shows that the query times for *getIndividualBelongingToClass* are higher than *getIndividualTypes*

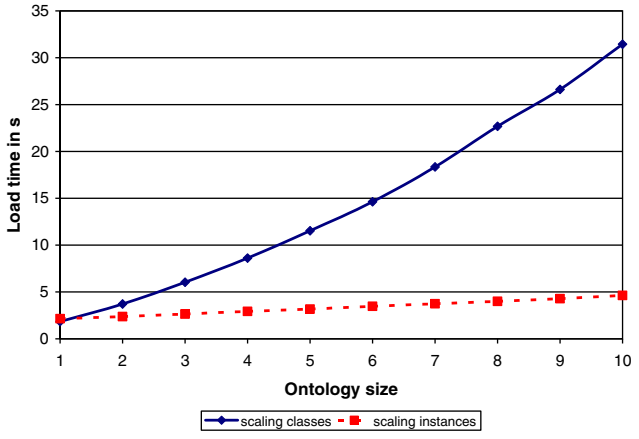

Fig. 4. Load time of ConceptBase for scaling of classes and instances.

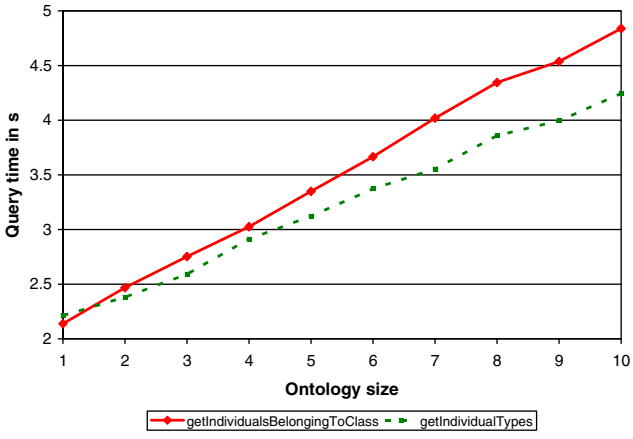

Fig. 7. Query time of Racer for scaling of instances.

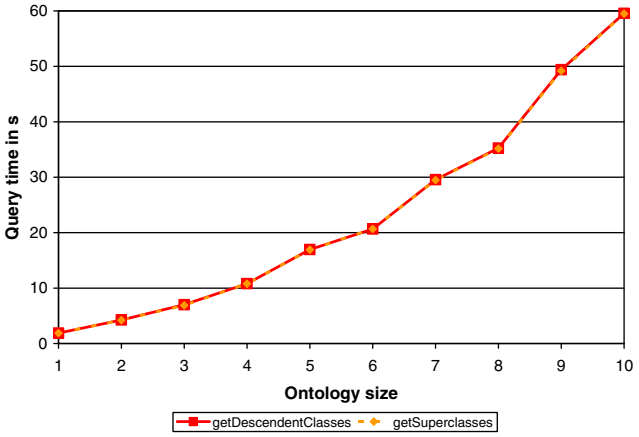

Fig. 5. Query time of Racer for scaling of classes.

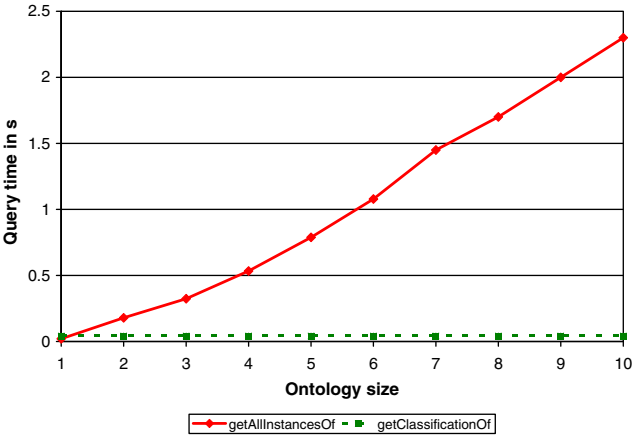

Fig. 8. Query time of ConceptBase for scaling of instances.

Please cite this article in press as: S.A. Ludwig, Comparison of a Deductive Database with a Semantic Web reasoning engine, Knowl. Based Syst. (2010), doi:10.1016/j.knosys.2010.04.005

Table 4
Summary of comparison criteria of ConceptBase and Racer.

Criteria	ConceptBase	Protégé/Racer
Documentation	Tutorial documents good regarding explicit how-to, but not enough depth User manual had sufficient depth but few examples	Good tutorials and documentation for Protégé and Racer
Feasibility and expressiveness	Allows for creation of subclasses and instances of classes, but also subclasses and instances of instances Does not allow properties to be defined outside of a class	Open world assumption; restriction must be explicitly stated No support for a numerical data type for the range or domain of properties
Complexity and usability	Difficult to remove base concepts; no DROP CASCADE function Graph editor allows data manipulation Restriction of main memory database (DB should not be larger than main memory of computer)	Graph editor (OWLViz) is only for viewing Many extra plugins for different views
Distribution	Client/server application KB needs to reside on server machine	Distributed computing paradigm KB can reside anywhere
Performance and scalability	Load time: linear for instance, quadratic for class scaling Scaling of classes: linear distribution Scaling of instances: linear distribution	Load time: linear for class and instance scaling Scaling of classes: quadratic distribution Scaling of instances: linear distribution

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 Fig. 8 the instance queries performed in ConceptBase are shown. The *getAllInstancesOf* query shows a quadratic distribution, whereas the *getClassificationOf* query shows a linear gradient. *getAllInstancesOf* takes longer as the return values range between 47 and 256 for ontology size 1–10, 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.

3.6. Discussion of findings

Table 4 shows a summary of the comparison criteria for both ConceptBase and Protégé/Racer. Regarding the documentation, Protégé provides better tutorials and documentation to help with the usage of the software package. Looking at feasibility and expressiveness – due to the open world assumption in Protégé, restrictions have to be explicitly stated, whereas this is not necessary in ConceptBase. Furthermore, Protégé does not allow for support of a numerical data type, which ConceptBase does.

Regarding the complexity and usability, both systems provide a graph editor, which makes the implementation of the application easier. One drawback of ConceptBase is that there is no option to remove base concepts and their corresponding subconcepts.

Looking at the criteria of distribution, ConceptBase is implemented as a client/server implementation. Therefore, the knowledge base resides on the server side. The Semantic Web technology including Protégé and Racer were built with the distributed computing paradigm in mind and thus, the knowledge bases can reside on different machines.

The direct comparison of queries revealed that the 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 where 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 larger amount of result values are returned, the performance decreases in ConceptBase, whereas Racer is not affected by this.

4. 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 a fair comparison could be performed.

The findings revealed that the reasoning capabilities in Racer are richer, with many redundant queries, as due to the usage of Datalog in ConceptBase certain problems are undecidable.

The performance and scalability analysis revealed that the load time to load the different ontologies was shorter in Racer, especially for the scaling of classes; the load time to load the different ontologies was smaller 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. However, the 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.

Considering that ontologies are developed incrementally, adding a relatively small increment to a large ontology has a great effect for loading 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.

Acknowledgements

The author would like to thank the Natural Sciences and Engineering Research Council of Canada (NSERC) for partial funding of this project. Furthermore, gratefully acknowledged are Christoph Quix and Manfred Jeusfeld for their helpful comments

on ConceptBase, and Michael Wessel for his valuable feedback regarding Racer.

References

[1] E.H. Shortliffe, MYCIN: Computer-Based Medical Consultations, Elsevier Press, New York, 1976.
[2] CLIPS Website, <http://www.ghg.net/clips/CLIPS.html> (retrieved November, 2009).
[3] JESS Website, <http://herzberg.ca.sandia.gov/jess/> (retrieved November, 2009).
[4] C.L. Forgy, Rete: a fast algorithm for the many pattern/many object pattern match problem, Artificial Intelligence 19 (1982) 17–37.
[5] A.T. Schreiber, B.J. Wielinga, R. de Hoog, H. Akkermans, W. van de Velde, CommonKADS: a comprehensive methodology for KBS development, IEEE Expert 28–37 (1994).
[6] J. Angele, D. Fensel, R. Studer, Developing knowledge-based systems with MIKE, Journal of Automated Software Engineering 5 (4) (1998) 389–418 (30).
[7] A. Voronkov, Lecture notes on Deductive Databases, <http://www.voronkov.com/dresden/2002/chapter_1.ps>, 2002 (retrieved November, 2009, Chapter 1).
[8] T.R. Gruber, Toward principles for the design of ontologies used for knowledge sharing, Special issue on the role of formal ontology in the information technology in International Journal of Human–Computer Studies 43 (5–6) (1993) 907–928.
[9] M. Jeusfeld, M. Jarke, From relational to object-oriented integrity simplification, in: M. Kifer, C. Delobel, Y. Masunaga (Eds.), Proceedings of the Deductive and Object-Oriented Databases, vol. 91, Springer-Verlag, 1991.
[10] M. Jeusfeld, M. Staudt, Query optimization in deductive object bases, in: G. Vossen, J.C. Freytag, D. Maier (Eds.), Query Processing for Advanced Database Applications, Morgan-Kaufmann, 1993.
[11] Protégé Website, <http://protégé.stanford.edu> (retrieved November, 2009).
[12] Racer Website, <http://www.racer-systems.com> (retrieved November, 2009).
[13] W. Nejdl, H. Dhraief, M. Wolpers, O-Telos-RDF: A resource description format with enhanced meta-modeling functionalities based on O-Telos, in: Proceedings of Workshop on Knowledge Markup and Semantic Annotation at the 1st International Conference on Knowledge Capture (K-CAP), Victoria, BC, Canada, October 2001.
[14] H. Knublauch et al., A Practical Guide To Building OWL Ontologies Using The Protégé-OWL Plugin and CO-ODE Tools Edition 1.0, Tutorial, 2004, <http://www.code.org/resources/tutorials/ProtegeOWLTutorial.pdf> (retrieved November, 2009).
[15] M.K. Mohania, N.L. Sarda, An architecture for a distributed deductive database system, in: Proceedings of IEEE Region 10 Conference on Computer, Communication, Control and Power Engineering, October 1993.
[16] M.K. Mohania, N.L. Sarda, Some issues in design of distributed deductive databases, in: Proceedings of the 20th International Conference on Very Large Data Bases, Santiago de Chile, Chile, September 1994.
[17] S. Lim, Y. Ng, Rule and data allocation in distributed deductive database systems, in: Proceedings of the 8th International Conference of Computing and Information (ICCI'96), Toronto, Canada, June 1996, pp. 979–1000.

Please cite this article in press as: S.A. Ludwig, Comparison of a Deductive Database with a Semantic Web reasoning engine, Knowl. Based Syst. (2010), doi:10.1016/j.knosys.2010.04.005