Abstract—With the increasing development of real applications using Semantic Web Technologies, it is necessary to provide scalable and efficient ontology querying and reasoning systems. In this paper we present DBOWL, a persistent and scalable OWL reasoner. Ontologies are stored in a relational database, using a description logic reasoner to pre-compute the class and property hierarchies and to obtain all the ontology information (i.e. properties domain and range) which is also stored in the database. Furthermore, a simple but expressive query language has been implemented, which allows us to query and reason on these ontologies. In order to show the use of the tool, we present an example that generates a relational database to store the Univ-Bench ontology. Finally, in order to assess the performance of our tool, we use LUMB, a well known benchmark to compare repositories in the Semantic Web extending it with more complex queries, and we compare the results obtained with Minerva. The results show the efficiency and scalability of DBOWL and the power of the query language.

I. INTRODUCTION

The OWL language [1] is being widely used to define ontologies in the Web. Its XML based syntax together with its correspondence with Description Logics (DL) [2], make it a good candidate to be the standard language for defining ontologies used by Semantic Web applications. However, there are still relatively few tools that allow us to manipulate, store and query ontologies defined using this language. Furthermore, Semantic Web applications, such as biological tools, use large ontologies, that is, ontologies with a large number (millions) of instances. Description logic based tools, like FaCT [3] or RACER [4] allow us to manage OWL ontologies, but not very large ones. Reasoning algorithms are not scalable and are usually main memory oriented. These reasoners are highly optimized for reasoning on the ontology structure (Tbox reasoning in Description Logic nomenclature), but have problems when dealing with reasoning on instances (Abox reasoning in Description Logic nomenclature). It is logical to think that applications in the Semantic Web will need to infer new knowledge from the explicit knowledge defined not only in the Tbox but especially in the Abox. The complex reasonings that should be implemented for the Semantic Web applications will need an optimal storage model in order to be efficient and scalable. This storage model will be disk oriented and not main memory oriented in order for it to be scalable. The database research community has successfully developed a wide theoretical corpus and mature and efficient technology to deal with large amounts of persistent information. Due to the size of the Semantic Web ontologies (very large ontologies with a very huge number of instances), we believe it will be necessary to use database technology in order to provide persistence to the knowledge described by the ontologies, as well as scalability to the queries and reasoning on this knowledge. Unfortunately, the application of database technology to reason with instances of OWL ontologies is not a trivial matter.

II. ARCHITECTURE OF DBOWL

DBOWL consists of two services, an OWL storage system (Figure 1) and an OWL querying and reasoning system (Figure 2). The OWL storage system stores the OWL ontology in the database. Starting from an OWL file, the class/subclass hierarchy (the concepts taxonomy), the property/subproperty hierarchy (the properties taxonomy), the ontology structure information and the ontology instances are computed using the description logic reasoner. Subsequently, a relational schema is created in order to store all this information. The relational schema is implemented using the Oracle database management system, and all the necessary information for implementing Tbox and Abox reasoning is them stored in the database. The DBOWL querying and reasoning system performs both Tbox reasoning and extended conjunctive queries over the ontology stored in the relational database. The DBOWL reasoner operates as follows. The user sends a query to the query processor. It can be either a Tbox reasoning or a conjunctive query. If it is a Tbox reasoning, the query planner sends the query to the Tbox reasoner which evaluates it using the database, and returns the result to the query planner. Otherwise, if the query is a conjunctive query, the query planner uses the Abox reasoner to infer all instances belonging to the classes and properties involved in the query which are not explicitly asserted in the ontology. The Abox reasoner evaluates the reasoning by accessing the database, and returns the result to the query planner. Finally, the query planner evaluates the query by translating it to SQL.

III. STORAGE, REASONING AND QUERYING

A. Storage in a Relational Database

Figure 3 shows the entity-relationship model for the database which stores the ontology. The ontology_index entity type has two attributes called seqnum and url, which store
the ontology identifier and the ontology name respectively. This entity type is weakly related to a weak entity type called element, which specializes in two entity types, namely class and properties. Properties also specialize in two entity types, object properties and datatype properties. For both class and properties (object and datatype ones) there will exist as many specializations as there are classes and properties in the ontology. Therefore, it is possible for each class to determine its instances using the corresponding relationship. An object property instance is a pair of class instances. A datatype property instance is a pair composed of one class instance and a value. We can also determine the instances of each property using the corresponding relationship. Each property has several attributes, namely transitive, symmetric and functional, which specify if the property fulfills these features, and domain and range which specify the domain and range of the property. Finally, the subclass_of and subprop_of relationships allow us to generate the classes and properties hierarchy of the ontology respectively and the equivalent to one stores the equivalent classes for each class.

B. Tbox and Abox Reasoning

DBOWL implements both Tbox and Abox reasoning. Tbox reasonings can be evaluated directly using the query language. On the other hand, Abox reasonings are evaluated when a query is sent to the system to obtain complete results. Currently, DBOWL supports all the Tbox reasonings implemented by RACER. In order to implement them, the information obtained from the DL reasoner is stored in the corresponding tables at load time. For example, we store in the database the equivalent classes for each class. Thus, we only need to query the database to evaluate the Tbox reasoning which evaluates if two classes are equivalent to each other or the equivalent classes of a specific class. We also use the DL reasoner to obtain the properties domain and range, which are sometimes not explicitly asserted by the ontology, but they can be inferred. At query time, this information will be obtained by querying the database with a simple SQL query. Obviously, the performance of these Tbox reasonings, being sound and complete, is much better than in a description logic reasoner, like FaCT [3] or RACER [4], which evaluate the reasoning each time in main memory.

Figure 4 shows the Abox reasoning currently supported by DBOWL. They are implemented as java functions using only the information stored in the database. We create views for each class and property in the ontology. These views store the instances of the corresponding class or property, i.e. instances explicitly asserted by the ontology plus instances inferred by the Abox reasonings. The views are created using functions that encapsulates a fix-point evaluation of each Abox reasoning. These views will be used in queries.

C. Queries

DBOWL evaluates Extended Conjunctive Queries (ECQ). An EQC is an expression of the form

\[ \text{ans}(V_1, ..., V_n) \leftarrow Q_1 \text{ AND } ... \text{ AND } Q_n. \]

Where:

- \( V_i \) is a set of variables that defines what we want to request.
- Every variable name begins with the symbol ?.
- \( Q_i \) can be \( C(x) \), \( P(x, y) \), \( C(x) \) OR \( D(x) \), \( \text{ALL } C(x) \), \( \leq n \ P(x, y) \), \( \geq n \ P(x, y) \), \( = n \ P(x, y) \).

And \( C, D \) are class names, \( P \) is a property name, \( x \) and \( y \) are instance names or variables, and \( n \) is a natural number.
An instance name can only appear in one of the variables of a property (never in both), and a substring search can be used by the symbol `%.

We define this syntax to be compatible with the most popular DL reasoner query languages syntax, such as nRQL [6], the RACER query language. EQC allows one to evaluate the same queries as nRQL plus databases-style queries, for example, queries with cardinality constraints. Furthermore, it supports disjunctive clauses using the OR operator.

An ECQ query is re-written before it is sent to the database for its evaluation. Each class or property name is substituted by the name of its corresponding view. This ensures the completeness of the results with respect to the implemented Abox reasonings. After that, the ECQ is translated to SQL.

IV. USE CASE: STORING, REASONING AND QUERYING THE UNIV-BENCH ONTOLOGY

In order to illustrate the use of the tool, we present an example that generates a relational database to store the Univ-Bench ontology [5] and we show some examples of Tbox reasonings and of ECQ queries that use Abox reasonings. The Univ-Bench ontology describes universities and departments and the activities that go on in them. University staff and students are classified using a taxonomy according to their category and their level (graduate or undergraduate) respectively. This ontology contains a large number of instances which are automatically generated. Therefore, it is a suitable ontology for highlighting the advantages of the persistent storage for this kind of tool.

A. Univ-bench ontology storage

As we can see in 5, the url_index table stores all ontology instances, giving a different ID from the URI of the instance for each one. For each class a table is created. The name of the table is the same as the class name plus the SEQNUM of the ontology which the class belong to. Examples of these tables are STUDENT_1 and FACULTY_1. In the same way, a table for each property is created. Examples of these tables are TEACHEROF_1 and TAKESCOURSE_1. Note that for tables representing properties we must store the two ID of the instances which are related.

B. Reasoning and Querying univ-bench ontology

As we mentioned before, our query language allows users to perform Tbox reasonings and ECQ directly in an intuitive way.
Below we present examples of Tbox reasonings with some queries and their results. We omitted the specific instances that queries return for space reasons. We check the completeness of these result using RACER. Therefore, DBOWL is currently as complete as RACER. Figure 6 and figure 7 show how to evaluate $R_1$ and $Q_3$ using the DBOWL graphical interface respectively.

**Tbox reasonings:**

- $R_1$: Subclasses of professor. Subclasses? $<$ −Professor
- Results: AssistantProfessor, AssociatedProfessor, Chair, Dean, FullProfessor, VisitingProfessor.
- $R_2$: Is dog an univ-bench class?. Class? $<$ −Dog
- Results: Dog is not a class
- $R_3$: Domain of the property teacher_of.
- Domain? $<$ −teacher_of
- Results: Faculty

**Queries:**

- $Q_1$: All professors
  \[ \text{ans}(?x)< -\text{professor(?x)} \]
  Results : 34 instances found.
- $Q_2$: All professors being full professors or assistant Professors
  \[ \text{ans}(?x)< -\text{professor(?x)} \text{ and fullprofessor(?x)} \text{ or assistantprofessor(?x)} \]
  Results : 20 instances found.
- $Q_3$: All professors being assistant professor teaching more than 3 subjects.
  \[ \text{ans}(?x,?y)< -\text{professor(?x)} \text{ and assistantprofessor(?x)} \text{ and teacherof(?x,?y)} \geq 3 \]
  Results : 25 instances found.

Users can perform queries combining all clauses defined in section III.C. As we see, the result of the queries includes instances inferred by Abox reasonings as well as instances explicitly asserted. No instances of professor are explicitly asserted by the ontology. However, $Q_1$ returns 34 instances corresponding to instances of professor subclasses. On the other hand, our query language implements the disjunction of concepts, which is not supported by both DL reasoners and other similar tools (see next section). Finally, queries involving cardinality constraints are allowed. The use of constraints is not contradictory to the application of reasoning.

V. RELATED WORKS

Our objective is to implement an OWL reasoner. As OWL is based on DL, we must study DL reasoners. Of these, RACER [4] is the most relevant and one of the most complete, and implements both Tbox and Abox reasoning. Furthermore, it provides its own query language, NRQL [6] which allows simple conjunctive queries to be evaluated. It is not persistent however, and reasoning is implemented by reducing it to satisfiability. This means on the one hand, that each time we use the reasoner, we must load and process the ontology and, on the other hand, that large ontologies (with large number of instances) cannot be loaded. Finally, RACER is currently a commercial tool, and therefore, other DL reasoners, like PELLET [7] are becoming more popular. PELLET provides the same functionality as RACER but also has the same problems.

In the past few years there has been a growing interest in the development of systems for storing large amounts of knowledge in the Semantic Web. Firstly, these systems were oriented to RDF storage [8] [9] [10] [11]. Nowadays, research is oriented to massive OWL storage. Several alternative approaches using relational technology have been presented. Instance Store [12] uses a DL reasoner for inferring Tbox information and storing it in a relational database. However, the ontology definition language does not allow the definition of binary relationships. From our point of view, this is an important expressiveness limitation. Moreover, Instance Store only evaluates a few Abox reasoning, namely subsumption of concepts and equivalent classes. It implements them by reducing them to terminological reasonings and evaluates them using a DL reasoner. On the other hand, QuONTO [13] system reduce the ontology definition language to DL-
Lite [14], a description logic which is a subset of OWL-DL. Therefore, the soundness and completeness of the reasonings is ensured. It evaluates subsumption of concepts Abox reasoning and conjunctive queries. The queries are rewritten using the Tbox information and they are translated to SQL. DLDB-OWL [15] extends a relational database with OWL inferences. This proposal uses a DL reasoner as Instance Store but the database schema is more complex. In its public distribution only the subsumption of concepts is implemented, but it is implementing using only the information stored in the database. Finally, Minerva [16] also stores the ontology in a relational database, but use a DL reasoner for evaluating Tbox reasonings and a rule engine to evaluating Abox reasonings which are defined using DLP [17] rules. This is, it combines relational technology with logic rules. Minerva also evaluates SPARQL [18] queries. Our proposal aims to combine all these results, providing a persistent and scalable tool for querying and reasoning on OWL ontologies. To do this, we provide an optimized storage model which is efficient and scalable, we implement reasoning on top of a relational database and combine reasoning and querying.

VI. DISCUSSION

In order to know the performance of our tool, we use LUMB, a well known benchmark to compare repositories in the Semantic Web [19]. This benchmark is intended to evaluate the performance of OWL repositories with respect to extensional queries over a large data set that commits to a single realistic ontology. Furthermore, the benchmark evaluates the system completeness and soundness with respect to the queries defined. This first experiment is conducted on a PC with Pentium IV CPU of 2.13 GHz and 2G memory, running Windows XP service pack 2 with Java JRE 1.4.2 Release 16. Two 1 and 10 Mb ontologies have been generated, in order to see the scalability.

We also evaluated Minerva [16] because it is the most complete of all the related tools. However, Minerva uses a logic engine in order to evaluated Abox reasoning. Logic engines are really difficult to implement and usually they need many optimization techniques in order to be efficient. For this reason the results of Minerva are not good (for some queries need 16 hours to obtain a result, and others do not give any result even after 24 hours using the 10 Mb ontology). Furthermore, Minerva does not support the cardinality restriction on queries. This is a common problem also for DL reasoners. Although Minerva supports SPARQL queries, not all SPARQL features are implemented. The DBOWL query language is at least as expressive as complete SPARQL. On the other hand, the results obtained evaluating DBOWL are not the final performance results due the DBOWL not being totally finished. We need to implement the rest of the Abox reasoning in order to get a good comparison, but the current results suggest that we are working in the right direction. Our highest response time is 26 milliseconds. However, our results are not complete because not all inferable instances are obtained.

Finally, current DL reasoners as well as related tools such as Minerva are monolithic system. This means that they apply all reasonings even though they are not necessary for solving the query. The results prove that if we can establish the reasonings which are necessary for solving a query a priori, and only these reasonings are applied, the performance time of the query can be drastically reduced. Some queries defined by the benchmark only use the subsumption of classes and properties reasonings. For this reason our tool evaluated them using much less time. Nevertheless, we are aware that it is very difficult to decide which reasonings are needed to solve a specific query, but we think that this is a new and really interesting area of research. If it were possible to define specific query languages which implement only those reasonings needed for a specific application, the process would be much less complex.

VII. CONCLUSIONS AND FUTURE WORKS

This paper presents DBOWL, a tool for querying and reasoning on OWL ontologies. It stores the ontologies in a relational database, using a description logic reasoner for pre-computing the class and property hierarchies, which are also stored in the database. Furthermore, a simple query language has been implemented, which allows us to query and reason on these ontologies. DBOWL supports both Tbox and Abox reasoning and Extended Conjunctive Queries (ECQ). This query language allows one to perform more complex queries than those allowed by DL reasoners and related tools. Reasonings are encapsulated by java functions making possible to configure the tool according to the reasoning needs of the applications. In order to prove our proposal we use a benchmark and we compare the results with Minerva. The results obtained suggest that DBOWL is a promising OWL reasoner. Currently, DBOWL supports much bigger ontologies than traditional DL systems. This is especially important for some applications, for example biological applications, where particularly large ontologies are used, like the Gene Ontology (http://www.geneontology.org/) or TAMBIS (http://www.ontologos.org), and in some research projects such as the Spanish government research project ICARO (TIN2005-09098-C05-01) or the Andalusian government funded Amine System Project (ASP CVI-657).

As future work, we plan to implement the rest of the Abox reasonings. The most important objective is the implementation of these Abox reasonings assuming the Abox stored in the database and exploiting its query capabilities.

ACKNOWLEDGMENT

This work has been supported by the Spanish MEC Grant (TIN2005-09098-C05-01).

REFERENCES


