ECQ: A Simple Query Language for the Semantic Web

Maria del Mar Roldan-Garcia, Joaquin J. Molina-Castro, Jose F. Aldana-Montes
Departamento de Lenguajes y Ciencias de la Computacion
Universidad de Malaga
Spain
Email: {mmar,jmolina,jfam}@lcc.uma.es

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 ECQ (Extended Conjunctive Queries), a simple but powerful query language for OWL. ECQ supports Tbox reasoning and complex queries, going beyond conjunctive queries, which are needed for real semantic web applications. The syntax of ECQ is the same as other description logic languages, making it easy to use for this community of researchers. We also present an implementation of the language over a scalable and persistent OWL reasoner, that is called DBOWL. Therefore, queries can use Abox reasoning to obtain more complete results.

I. INTRODUCTION

The OWL language [1] is being widely used to define ontologies in the Web. This language provides three increasingly expressive sublanguages, namely OWL Lite, OWL-DL and OWL Full. The OWL-DL 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-DL ontologies, but not very large ontologies. Reasoning algorithms are not scalable and are usually 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 a very large number (millions) of instances (Abox reasoning in Description Logic nomenclature) and with queries. They perform at most conjunctive queries. It is logical to think that applications on the Semantic Web will need to infer new knowledge from the explicit knowledge defined not only in the Tbox but especially in the Abox. Besides, these applications will need to evaluate more complex queries than conjunctive queries. In order to do this, a query language is needed. This query language must be based on description logic and must be able to evaluate Tbox reasoning and complex queries. Furthermore, in order to obtain complete results, queries must use the results of Abox inferences supported by OWL-DL. In this paper we present ECQ (Extended Conjunctive Queries), a simple but powerful query language for the semantic web which presents all these features. ECQ has the same syntax as description logic query languages enabling description logic researchers to use it. We also present an implementation of the language over a scalable and persistent OWL-DL reasoner, that is called DBOWL [5]. DBOWL supports storage, querying and reasoning of OWL-DL ontologies. From this point onwards in the paper we use only OWL to refer to OWL-DL.

II. DESCRIPTION OF THE QUERY LANGUAGE

ECQ allows performing Tbox reasoning and extended conjunctive queries over an OWL ontology. Besides, Abox reasoners are evaluated when a query is sent to the query processor to obtain complete results. An ECQ is an expression of the form

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

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 \( C(x), P(x, y), C(x) \mathbf{OR} D(x), \mathbf{ALL} C(x), \mathbf{<=} n\ P(x, y), \mathbf{>=} n\ P(x, y), = n\ P(x, y) \).
- 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. ECQ allows users 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.

III. IMPLEMENTATION OF THE QUERY LANGUAGE

We implement the query language using a relational database to store the OWL ontologies. The Abox reasoning used by the queries is also implemented on top of this database. The complete system is called DBOWL\(^1\) [5].

\(^1\)http://khaos.uma.es/dbowl
A. 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 then stored in the database. The DBOWL querying and reasoning system performs both Tbox reasoning and ECQ 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 an extended 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 extended 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.

B. Storage of the ontology 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 fulfils 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 stores the equivalent classes for each class.

C. Tbox and Abox Reasoning implementation

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, in the database we store 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 if a class is 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 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
Fig. 3: EER schema of the database storing the OWL ontologies

reasoner, like FaCT [3] or RACER [4], which evaluate the reasoning each time in the 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 encapsulate a fix-point evaluation of each Abox reasoning. These views will be used in queries.

D. ECQ Queries implementation

An ECQ query is re-written before it is sent to the database for 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 using a specific algorithm. This algorithm produces a translation from an ECQ query to an efficient SQL query, so that users can perform complex SQL queries using a simply logical language. This translation is carried out following three steps (the detailed algorithm is shown at the DBOWL web site):

1) For each variable in the query, the names of the views (views of classes and properties) in the query body which use that variable, and the logical operators (AND or OR) which related these view names, are determined. These associations are used to create an initial SQL query that contains the SELECT fields to determine which columns have to be requested, the FROM fields to determine which views have to be used in the query, the necessary WHERE clauses to determine which column views have to be associated, and the ORDER BY field to determine how to order the result.

2) For each constant, the constant restriction clauses that have to be added to the WHERE clauses for the correct translation of the query are determined.

3) Finally, for each condition (ALL, >=, <=, =), the condition restriction clauses that have to be added to the WHERE clauses to obtain the final SQL query, are also determined.

The next detailed example shows the complexity of the queries generated by this algorithm.

The ECQ query,

\[ \text{ans}(?x,?y) \leftarrow \text{fullprofessor}(?x) \text{ or assistantprofessor}(?x) \text{ and worksfor}(?x,\%\text{university%}) \text{ and } \geq 3 \text{ teacherof}(?x,?y) \text{ and ALL course(?y)} \]

Is processed step by step obtaining:

1) After associated ?x: select distinct u1.url from uri_index u1, worksfor_l_p w1, teacherof_l_p t1, course_l_c cl where url.id=w1.subject and w1.subject=t1.subject and
users to perform Tbox reasonings and ECQ directly in a kind of tool. As mentioned before, our query language allows automatically generated. Therefore, it is a suitable ontology contains a large number of instances which are other hand, our query language implements the disjunction corresponding to instances of professor subclasses. On the asserted by the ontology. However, Q explicit asserted. No instances of professor are explicitly instances inferred by Abox reasonings as well as instances section III.C. As we see, the result of the queries includes 20 instances found. 

**Queries:**

**Q1:** All professors ans(?x) ← professor(?x)

**Results:** 34 instances found.

**Q2:** All professors being full professors or assistant professors ans(?x) ← professor(?x) and fullprofessor(?x) or assistantprofessor(?x)

**Results:** 20 instances found.

**Q3:** All professors being assistant professor teaching more than 3 subjects.

ans(?x,?y) ← professor(?x) and assistantprofessor(?x) and >= 3 teacherof(?x,?y)

**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, Q1 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.

IV. USE CASE: QUERYING AND REASONING ON UNIV-BENCH ONTOLOGY

In this section we show some examples of Tbox reasonings and of ECQ queries that use Abox reasonings. The Univ-Bench ontology [7] 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 persistent storage for this kind of tool. As 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 results using RACER. Therefore, DBOWL is currently as complete as RACER. The reader can test a DBOWL demo at its web site.

**Tbox reasonings:**

**R1:** Subclasses of professor. Subclasses? ← Professor

**Results:** AssistantProfessor, AssociatedProfessor, Chair, Dean, FullProfessor, VisitingProfessor.

**R2:** Is dog a univ-bench class?. Class? ← Dog

**Results:** Dog is not a class

**R3:** Domain of the property teacher_of.

Domain? ←teacher_of

**Results:** Faculty

<table>
<thead>
<tr>
<th>SubClassOf</th>
<th>If x is instance of C and C ⊑ D, then x is instance of D.</th>
</tr>
</thead>
<tbody>
<tr>
<td>EquivalentClass</td>
<td>If x is instance of C and C ⊑ D, then x is instance of D.</td>
</tr>
<tr>
<td>SubPropOf</td>
<td>If (x,y) is instance of P and P ⊑ Q, then (x,y) is instance of Q.</td>
</tr>
<tr>
<td>Transitivity</td>
<td>If (x,y) is instance of P and P is transitive, then (x,y) is instance of P.</td>
</tr>
<tr>
<td>Symmetry</td>
<td>If (x,y) is instance of P and P is symmetric, then (x,y) is instance of P.</td>
</tr>
<tr>
<td>Inversion</td>
<td>If (x,y) is instance of P and Q is the inverse property of P, then (y,x) is instance of Q.</td>
</tr>
<tr>
<td>Domain</td>
<td>If (x,y) is instance of P, and C is the domain class of P, then x is instance of C.</td>
</tr>
<tr>
<td>Range</td>
<td>If (x,y) is instance of P, and D is the range class of P, then x is instance of D.</td>
</tr>
</tbody>
</table>

Fig. 4: Abox Reasonings supported by DBOWL

<table>
<thead>
<tr>
<th></th>
<th>SubClassOf</th>
<th>If x is instance of C and C ⊑ D, then x is instance of D.</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>EquivalentClass</td>
<td>If x is instance of C and C ⊑ D, then x is instance of D.</td>
</tr>
<tr>
<td></td>
<td>SubPropOf</td>
<td>If (x,y) is instance of P and P ⊑ Q, then (x,y) is instance of Q.</td>
</tr>
<tr>
<td></td>
<td>Transitivity</td>
<td>If (x,y) is instance of P and P is transitive, then (x,y) is instance of P.</td>
</tr>
<tr>
<td></td>
<td>Symmetry</td>
<td>If (x,y) is instance of P and P is symmetric, then (x,y) is instance of P.</td>
</tr>
<tr>
<td></td>
<td>Inversion</td>
<td>If (x,y) is instance of P and Q is the inverse property of P, then (y,x) is instance of Q.</td>
</tr>
<tr>
<td></td>
<td>Domain</td>
<td>If (x,y) is instance of P, and C is the domain class of P, then x is instance of C.</td>
</tr>
<tr>
<td></td>
<td>Range</td>
<td>If (x,y) is instance of P, and D is the range class of P, then x is instance of D.</td>
</tr>
</tbody>
</table>
V. RELATED WORKS

ECQ is an OWL query language. As OWL is based on DL, we must study DL reasoners query languages. Of these, nRQL [6], which is the RACER [4] query language, allows evaluating simple conjunctive queries. RACER is the most relevant DL reasoner, and one of the most complete, and implements both Tbox and Abox reasoning. However, it is not persistent, 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 [8], are becoming more popular. PELLET query language provides the same functionality as nRQL, but also has the same problems.

Recently, the W3C consortium proposed SPARQL [9] as a standard Semantic Web query language. SPARQL allows us to query RDF, and its syntax is similar to SQL syntax. Therefore, it is not an easy language. People not familiarized with SQL might have problems writing SPARQL queries. The semantics of SPARQL is similar to ECQ, except for the disjunction of classes, that is only supported by ECQ. However, there is still no a complete implementation of this language. ECQ is completely implemented and queries are easy to write, as we showed in the section above. Furthermore, since ECQ syntax is the same as the description logic query languages syntax, people used to working with description logics reasoners easily use this language. Other RDF query languages [10] such as RQL [11] or RDQL [12] present the same problems described for SPARQL.

Finally, in the context of the DL DL-Lite [13], a theoretical study is carried on about which Abox reasonings can be implemented using SQL statements. However, DL-Lite is not a very expressive DL and we think that real applications in the Semantic Web will need more expressiveness than that offered by DL-Lite. We also believe that the use of SQL-views for implementing Abox reasoning is a good approach and at the moment we are obtaining good results. Currently we are working on the theoretical demonstration of our approach. DL-Lite also defines a query language which supports conjunctive queries. ECQ extends conjunctive queries with disjunction and other operators. Therefore, ECQ allows more complex queries than simple conjunctive queries.

VI. CONCLUSIONS AND FUTURE WORKS

This paper presents ECQ (Extended Conjunctive Queries), a simple but powerful query language for the semantic web. ECQ has the same syntax as description logic query languages, enabling description logic researchers to use it. This language allows users to perform more complex queries than those allowed by DL reasoners, as well as Tbox reasoning. Moreover, we present an implementation of ECQ over a persistent and scalable OWL reasoner called DBOWL. 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. Abox reasonings are encapsulated by Java functions, making it possible to configure the tool according to the reasoning needs of the applications. Queries uses this reasoning to obtain more complete results. The current DBOWL implementation is available at its web site, http://www.khaos.uma.es/dbowl.

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. We are also carrying out a study of the performance of the ECQ queries with the objective of comparing it with others query languages. Finally, we are working on two important theoretical issues. On one hand, a formal proof that for every Abox reasoning, DBOWL returns the correct set of results. On the other hand, a formal proof that for every ECQ query, the SQL query produced by our translation algorithm also returns the correct set of results.

ACKNOWLEDGMENT

This work has been supported by Grants ICARO Project, TIN2005-09098-C05-01 (Spanish Ministry of Education and Science), and Applied Systems Biology Project, P07-TIC-02978 (Junta de Andaluca).

REFERENCES