The TOQL system

Evdoxios Baratis[1], Nikolaos Maris[1], Euripides G.M. Petrakis[1], Sotiris Batsakis[1], and Nikolaos Papadakis[1]

Department of Electronic and Computer Engineering
Technical University of Crete (TUC)
Chania, Greece
e-mail:{dakis, petrakis}@intelligence.tuc.gr, nickmeet@gmail.com, {batsakis, npapadak}@intelligence.tuc.gr

Abstract. TOQL, is a query language for querying time information in ontologies. An application has been developed that supports translation and execution of TOQL queries on temporal ontologies. A Graphical User Interface (GUI) has been also developed to facilitate user interaction and supports operations such as syntax highlighting, code autosuggestion, loading of the ontology into the main memory, results and error display.

1 Introduction

TOQL (Temporal Ontology Querying Language), is a high-level query language for querying (time) information in ontologies. TOQL handles ontologies almost like relational databases. TOQL maintains the basic structure of an SQL language (SELECT - FROM - WHERE) and treats the classes and the properties of an ontology almost like tables and columns of a database. The following table summarizes TOQL syntax:

Syntax
SELECT ... AS ...
FROM ... AS ...
WHERE ... LIKE ... AND ... LIKE “string” IGNORE CASE ... AT...

Table 1. Generic TOQL syntax.

The TQQL system supports query translation and execution of temporal queries (i.e. queries that contain the AT and Allen temporal operators) along with a mechanism for representing time evolving concepts in ontologies inspired by the four-dimensional perdurantist approach [4]. The 4D perdurantist mechanism is not part of the language and it is not visible to the user (so the user need not be familiar with peculiarities of the underlying mechanism for time information representation). A graphical user interface has also been developed to facilate user interaction with both TOQL and with the knowledge base (the termporal ontology).

2 System Architecture

The TOQL system has been implemented in Java. The system supports query translation and execution of TOQL queries on OWL ontologies containing temporal information.

Figure 1 illustrates the architecture of the TOQL system. The TOQL system consists of several modules, the most important of them being the TOQL interpreter whose purpose is to translate the TOQL query into a SeRQL [1] query, which is executed on the knowledge base. TOQL and SeRQL have different syntax and SeRQL does not support the full range of TOQL's time features. Part of the interpreter is a reasoner implemented in Prolog. The reasoner handles queries concerning properties that conform to the event calculus axioms [5]. The complete discussion of the TOQL implementation can be found in [2].

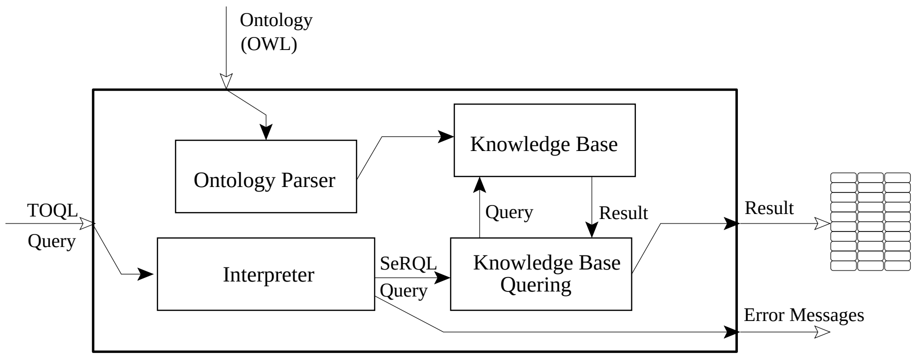


Fig. 1. TOQL system architecture.

2.1 Ontology Parser

The input is a query written in TOQL and an ontology in OWL. The ontology is parsed using JENA [1] and SESAME [2] and is loaded into the main memory. The ontology is checked for consistency with the 4D fluent mechanism [2] (error messages are reported to the output). The SeRQL query addresses both the ontology structure (TBOX) and the knowledge base instances (ABOX). Inferred facts are asserted in the KB using the Pellet[3] OWL reasoner. A query language alone can be used to access temporal information that is explicitly represented in a temporal ontology, but cannot provide answers on information that can be inferred from existing information (e.g., if the price of a product at time t is p, TOQL should be able to infer that the price of the product is the same

[1] http://sourceforge.net/projects/jena
[2] http://www.openrdf.org
[3] http://clarkparsia.com/pellet

since the last time it was changed). TOQL is combined with a reasoner based on event calculus [5] to better support queries on temporal ontologies. The output to a query is a table with the results. If errors have been encountered during interpretation, the output is one or more error messages.

2.2 Interpreter

A TOQL query is initially lexically, syntactically and semantically analyzed. Lexical analysis converts a sequence of characters to tokens (i.e., meaningful labels). The lexical analysed is implemented using JFlex [4]. For example, the token SELECT is given the meaning SELECT, while the token "Linux" is given the meaning NAME. The next step is syntax analysis (parsing) whose purpose is to analyze a sequence of tokens to determine grammatical structure (i.e., allowable expressions) with respect to a given formal grammar [2]. The parser transforms the query to a syntax tree, a form suitable for further processing. The parser is implemented using Byacc/J [5]. Syntactical errors are reported in the output. If the query is lexically and syntactically correct, query translation proceeds with semantic analysis. Semantic analysis adds semantic information to the parse tree and builds the symbol table. This phase performs two types of semantic checks. The first type needs no external knowledge. Semantic errors reported in this case include, use of a class in a SELECT or WHERE clause without having it declared in the FROM clause, use of a property in a SELECT or WHERE clause without a class preceding it,and use of more than one properties in the SELECT clause of a nested query.

Detection of the second type of semantic errors needs external knowledge (i.e., the ontology). This requires that the ontology is first loaded into the main memory. The ontology is parsed using JENA and SESAME libraries. The semantic analyzer checks if a class or property used in a query exists in the ontology, if a property is a property of a specific class and finally, if a property is a fluent property (so that keyword TIME can be applied to it). A complete list of error messages that can be reported by the semantic checker can be found in [2].

Code generation: The last phase of query processing is the actual translation of a TOQL into an equivalent SeRQL query. Code generation performs the following steps:

- Intermediate code generation.
- Intermediate code parsing and instantiation to Java objects (representing the TOQL query).
- Processing of Java objects and expansion with 4D fluent elements.
- Processing of Java objects and mapping to Java objects (representing the SeRQL query).
- SeRQL query generation.

Finally the SeRQL query is applied to the Knowledge Base using SESAME and the result is presented to the user.

[4] http://jflex.de
[5] http://byaccj.sourceforge.net

3 Graphical User Interface

A Graphical User Interface (GUI) has been also developed to facilitate user interaction with TOQL. It supports operations such as syntax highlighting and loading of the ontology into the main memory. The toolbar panel provides buttons for query editing (undo, redo, copy, cut, paste), for displaying query results as well as for displaying the SeRQL equivalent query. Syntactic and semantic errors are also displayed. The interface, provides options for ontology loading (the "Load Ontology" button loads a new ontology into the memory) and ontology viewing (i.e., "View Ontology" button displays the Abstract Ontology View referred to [2], which hide the temporal mechanism representation from the user). The query editing panel contains the query editor and a toolbar panel that contains buttons useful for querying editing (save, save as, load query, run).

Query formulation is supported by *TOQL syntax highlighting* (recognizes TOQL clauses keywords and classes-properties) and by *Code autosuggestion* (each time the user writes a class name followed by ".", a list with the class properties is displayed to choose from).

The results panel has two tabs. The first one displays the results returned by the query, while the second one displays the errors returned as the result of query parsing. These errors can be either due to inconsistencies with the 4D fluent representation or due to errors in TOQL syntax.

4 Conclusions

The TOQL system provides a high level user interface to TOQL, an SQL-like language for querying temporal information in ontologies. TOQL is currently being extended to handle queries on ontology structure (i.e., sub- classes and super-classes) as well as open-schema query functionality by allowing variables in the class position (e.g. using rdf:type as a property in a query triple).

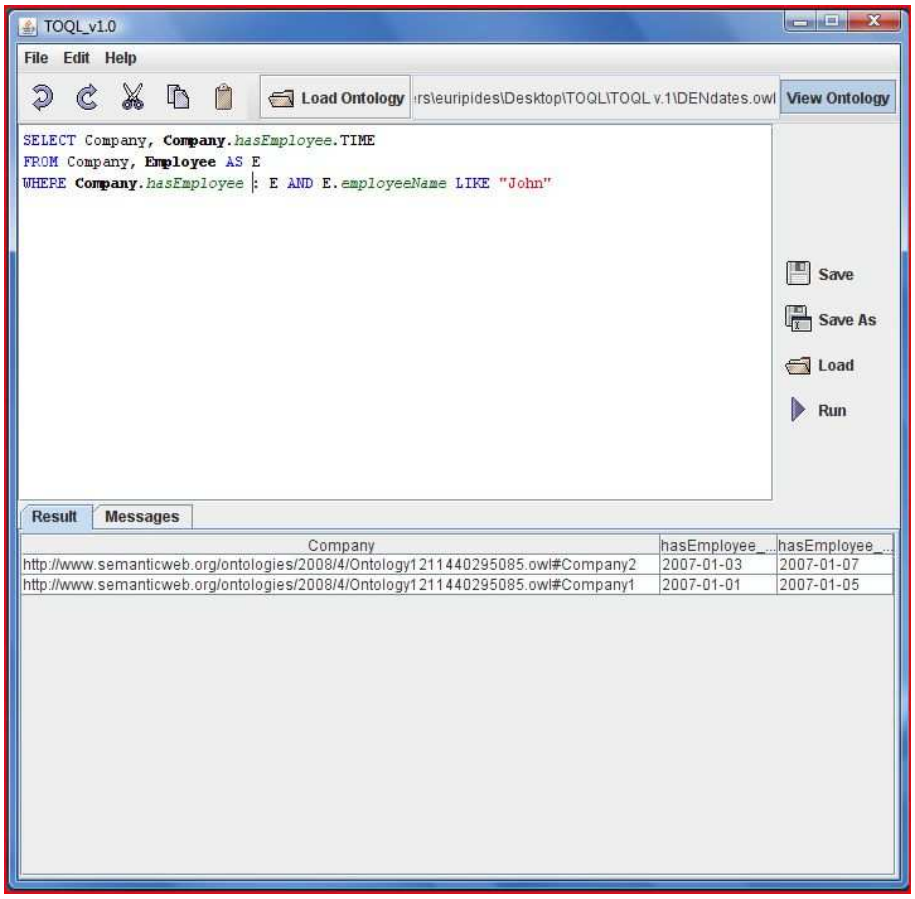
References

1. B. V. Aduna. The SeRQL query language. User Guide for Sesame 2.1, Chapter 9, 2002–2008. http://www.openrdf.org/doc/sesame2/2.1.2/users/ch09.html.
2. E. Baratis. TOQL: Querying Temporal Information in Ontologies. Master's thesis, Techn. Univ. of Crete (TUC), Dept. of Electronic and Comp. Engineering, July 2008.
3. D. L. McGuinness and F. VanHarmelen. OWL Web Ontology Language Overview. W3C Recommendation, February 2004. http://www.w3.org/TR/owl-features.
4. C. Welty, R. Fikes, and S. Makarios. A Reusable Ontology for Fluents in OWL. Technical Report RC23755 (Wo510-142), IBM Research Division, T. Watson Research Center, Yorktown Heights, NY, October 2005.
5. M. Shanahan. The event calculus explained. In Wooldridge, M., and Veloso, M.(Eds.), Artificial Intelligence Today, pp. 409430. Springer Lecture Notes in Artificial Intelligence no. 1600, 1999.

The TOQL System- Demonstration Outline

The TOQL sytem demonstration will include:

– Presentation of TOQL query examples.
– Presentation of the TOQL system interface and of its functionality (Figure 2).



```
SELECT Company, Company.hasEmployee.TIME
FROM Company, Employee AS E
WHERE Company.hasEmployee : E AND E.employeeName LIKE "John"
```

Company	hasEmployee_...	hasEmployee_...
http://www.semanticweb.org/ontologies/2008/4/Ontology1211440295085.owl#Company2	2007-01-03	2007-01-07
http://www.semanticweb.org/ontologies/2008/4/Ontology1211440295085.owl#Company1	2007-01-01	2007-01-05

Fig. 2. Graphical user interface: TOQL query with answer.

– Comparison between TOQL and SeRQL syntax demonstrating the rich expressive power and simpler syntax of TOQL.