

# Ontoligent Interactive Query Tool

Christopher J. O. Baker, Xiao Su, Greg Butler, and Volker Haarslev

Department of Computer Science & Software Engineering,  
Concordia University, Montreal, Quebec, Canada.  
<http://www.cs.concordia.ca/FungalWeb/>  
[baker@cs.concordia.ca](mailto:baker@cs.concordia.ca), [xu@cs.concordia.ca](mailto:xu@cs.concordia.ca),  
[gregb@cs.concordia.ca](mailto:gregb@cs.concordia.ca), [haarslev@cs.concordia.ca](mailto:haarslev@cs.concordia.ca)

**Abstract.** Recognizing the complexity of the Description Logic-based query of OWL-DL ontologies for the non-specialist, we introduce a query tool called Ontoligent Interactive Query (OntoIQ). This tool provides a well-organized user interface for a variety of users, from the beginner to the professional. Users can browse ontologies and build queries using query patterns and ontology content. OntoIQ translates queries automatically into the new RACER Query language (nRQL) [1] syntax and presents them to description logic automated reasoner RACER [2] which returns the query results. The tool includes import and export functions so that queries can be stored, shared, and re-imported by other users. Users are not required to learn the nRQL syntax. OntoIQ software is available for download at The FungalWeb website.

## 1 Introduction

Ontologies are now cornerstones of many knowledge discovery platforms and are integrated within information systems. In recent years much effort has been focused on the development of ontologies and the tools for their creation / edit. Despite this increase in the abundance of freely accessible ontologies a number of additional challenges remain. These include ontology visualization, for which some tools now exist. Tools such as the Protégé [3] plug-in TGVizTab [4], and Growl [5] are critical tools in the iterative development lifecycle of ontologies involving the ontology engineer, the domain expert and end user. The full value of OWL ontologies can only however be realized through the Description Logic (DL) paradigm and associated tools [2], [6], [7] which support reasoning over the ontological conceptualization to derive implicit knowledge. Appropriately, a plug-

in for Protégé makes it possible for users to pose DL-queries from the ontology editor [8]. Although DL-query languages facilitate the interrogation of ontologies, the Lisp based syntaxes such as that of nRQL and RACER are difficult for domain experts to master [9] particularly when seeking to evaluate ontologies for use in their own contexts. For this reason we sought to develop an interactive query tool called the Ontoligent Interactive Query tool (OntoIQ) which mirrors the basic query functionalities provided by nRQL used with RACER but with the advantage of browse and click operability.

### 2.1 nRQL as a Knowledge Representation Query Language

The query of knowledge representation formalisms such as ontologies is a central requirement of the Semantic Web. Since the recent establishment of the Ontology Web Language (OWL), design specifications for DL-based query languages have been proposed and existing languages contrasted, highlighting their advantages and limitations [10]. nRQL emerges as a prominent and highly expressive DL-query language. It extends the existing capabilities of the RACER with a series of query atoms, namely Unary concept query atoms, Binary role query atoms, Binary constraint query atoms, Binary same-as atoms, Unary has-known-successor atoms, and Negation atoms. nRQL uses a Lisp based syntax and the general structure of a query is composed of a query head i.e. Retrieve (?x) upon which variables used in the body are projected e.g. (?x Fungi), where (retrieve (?x) (?xFungi)), queries for instances of the concept Fungi. In this paper we illustrate conjunctive queries where the atoms are simple concept or role assertions where the variables in the body of the query match the corresponding individuals in the ontology that satisfy all query conditions. For some elements/operations of nRQL queries a closed world assumption is assumed, referred to also as 'active domain semantics' where only named individuals from the Abox will be considered as matches for query variables. A detailed description of nRQL is given in [11] and verbose examples outlined in [12].

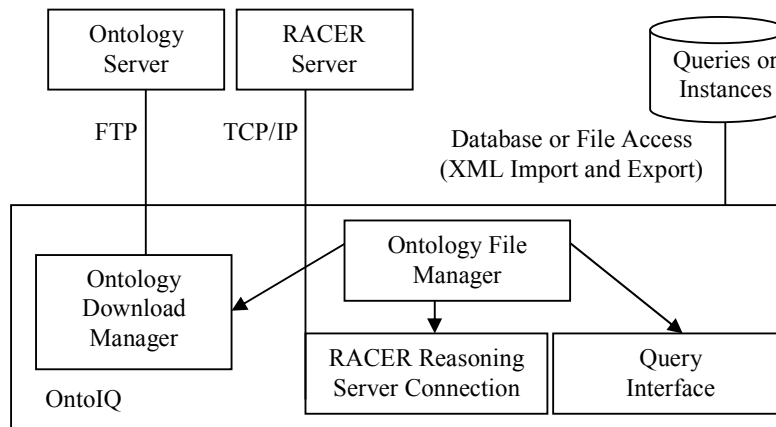
## 2 The Ontoligent Interactive Query Tool

OntoIQ was designed to give domain experts the opportunity to interface with OWL conceptualizations and interrogate them to assess their content using semantic web technology. In the next sections we introduce the OntoIQ system architecture, query patterns and query interface features of the tool.

### 2.1 System Architecture

We developed OntoIQ using a multi-tiered architecture shown in Fig. 1. This comprises of the following: an Ontology Download Manager, a RACER Server

Connection Interface, an Ontology File Manager, and a Query Interface. Screenshots of these interfaces are shown in Fig 2. The Ontology Download Manager allows for specification of server settings for the download of knowledge representation resources hosted locally on a user's machine or available elsewhere on the World Wide Web. The RACER Reasoning Server Connection interface allows specification of a remote connection to the RACER Server or to a local executable file of the reasoning engine. Where a remote connection with a RACER server is established the ontology file (OWL file) must be transported from the OntoIQ client. OntoIQ uses a socket on the TCP/IP protocol requiring the specification of the IP address of the remote RACER server. The remote RACER server must be running in the unsafe mode to permit upload and storage of the OWL file. Large ontology files greater than 5Mb have slower transfer speeds.



**Fig. 1.** System Architecture of the Ontoligent Interactive Query tool (OntoIQ).

The Ontology File Manager serves to load local ontology files to the RACER Server and to invoke the checking of the consistency and classification of the loaded ontology. It further permits the selection of six nRQL running modes [1]. The Query Interface provides the gateway to a series of query options. Direct access to the command line of the Reasoner is provided for advanced users wishing to pose queries using RACER syntax not supported by OntoIQ.

## 2.2 User Overview

To illustrate the interaction of the user with OntoIQ, we describe a typical workflow for a new user. The user downloads the OntoIQ tool and runs it from the local machine. An IP address for the remote Ontology Server is provided by the user to the Ontology Download Manager and an ontology file is downloaded.

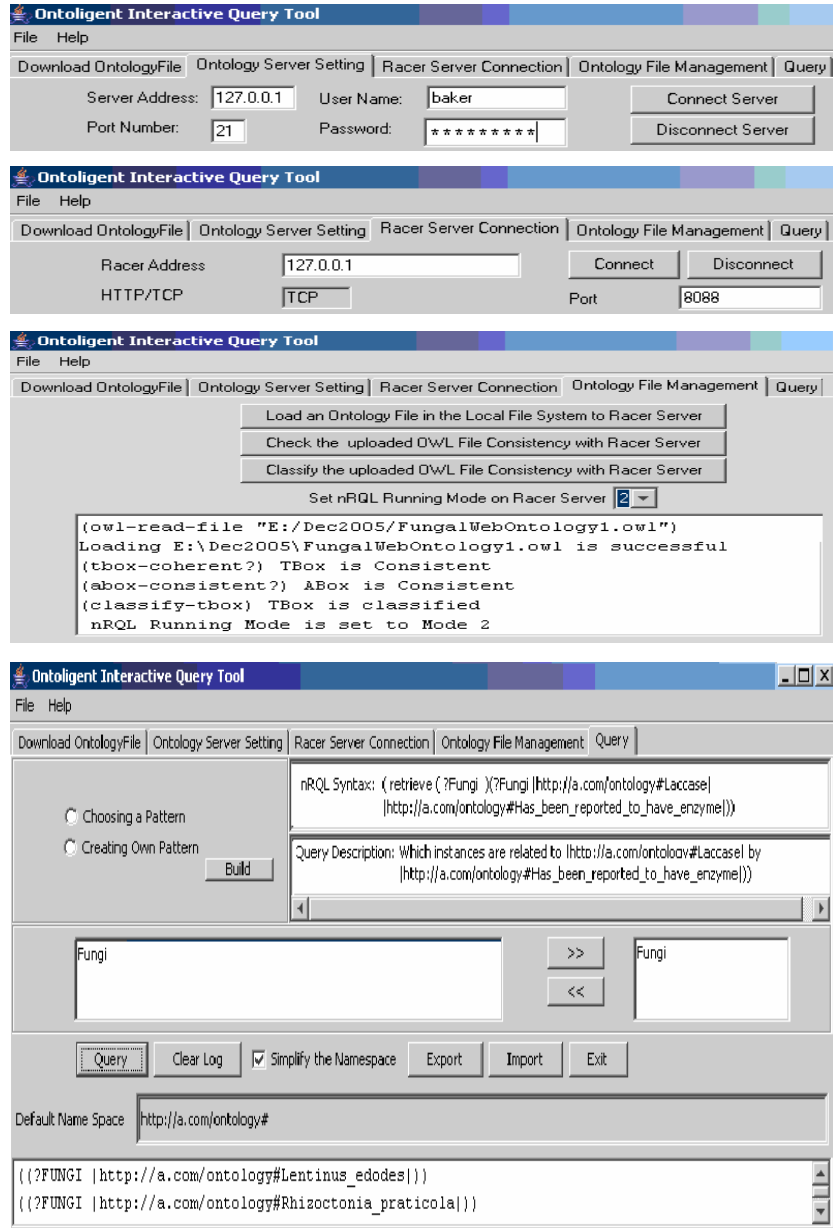


Fig. 2. Screenshots of OntoIQ User Interaction Panels

A connection is established between OntoIQ and a RACER server. An ontology file is loaded and a consistency and classification check on the ontology is run. The user then builds queries using the query interface selecting a query pattern and ontology specific content by browsing the ontology with OntoIQ. Queries are subsequently translated to nRQL syntax and sent to the RACER server. RACER returns the query results which can be saved along with the query syntax for export and reuse at a later time.

### 2.3 Query Patterns

A range of query patterns are available within OntoIQ. These are summarized in Table 1. The capabilities of the query patterns and the required information that must be provided by the user are outlined. Simple query patterns such as the *concept* pattern, the *role* pattern, and more complex patterns such as *intersect-conjunction*, *union-disjunction* and *combination* patterns can also be constructed.

**Table 1** Query patterns available in OntoIQ

Pattern	Syntax	Query
Concept	A concept and a variable	Is there any instance (not) belonging to the concept 'Enzyme'? What individuals (do not) belong to the concept 'Enzyme'? Does the instance Laccase (not) belong to the concept 'Enzyme'?
Role	A role and the role's domain and range.	What pairs of individuals are (not) related by role <i>has_been_reported_to_have_enzyme?</i> Is there any instance (not) related by the role <i>has_been_reported_to_have_enzyme?</i> To what individuals is the instance Laccase related by the role <i>has_been_reported_to_have_enzyme?</i>
Intersect / Conjunction	Simple concept or role patterns in combination	What pairs of individuals are (not) related by role <i>has_been_reported_to_have_enzyme</i> and are related to the concept 'Substrate' by the role <i>has_activity_towards_substrate</i>
Union / Disjunction	Simple concept or role patterns in combination	Retrieve individuals that belong to concept Hydrolase or individuals that belong to concept Lyase
Combo (And / Union)	Combines any combination of patterns.	Find individuals of the concept 'Commercial_Enzyme_Product' that are related by the role <i>contains</i> where the instance is 'xylanase'.

The process of building such queries through a browser requires a user to identify in advance the pattern required for their intended query. Thereafter the user

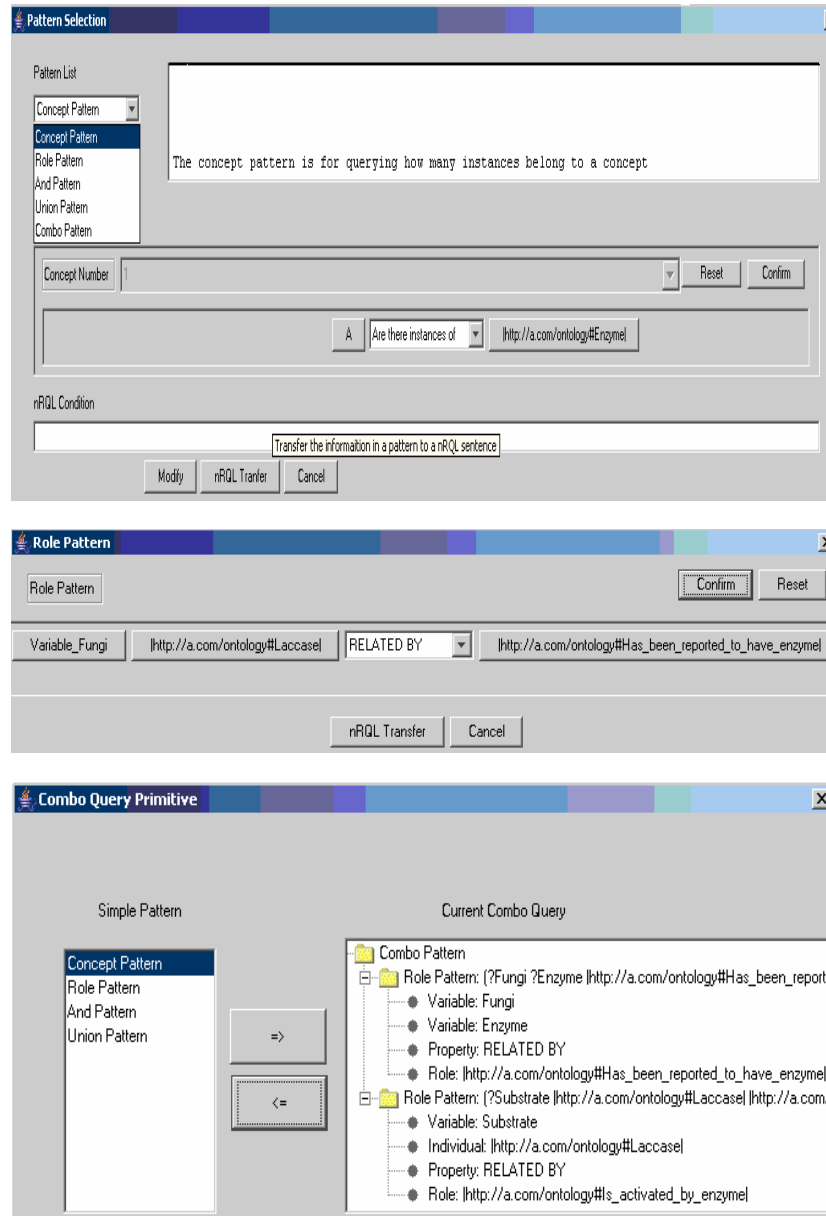
selects the concepts, roles, and individuals of the concepts or the relationships among these individuals from within the ontology.

## 2.4 Query Interface

The Query panel (Fig. 2) provides access to a cascade of windows, the first of which allows for the selection of query patterns using the ‘build’ button to access the Pattern Selection pane (Fig. 3). The ‘Pattern List’ menu of the Pattern Selection pane permits the selection of five pattern templates. Depending on the pattern selected a series of additional information must be provided. The Concept Pattern template shown requires a variable, i.e. a symbol representing the concept, or instance of the concept in the query syntax, along with the description of the concept or instance in the ontology i.e. its name obtained from browsing the ontology file. In Fig. 3 the OntoIQ query pattern templates are illustrated using the OWL file of the FungalWeb Ontology [13]. The Concept Pattern built selects for instances of the concept *Enzyme*. Each query pattern template can be built to specify affirmation or negation, i.e. presence or absence of instances using a drop box menu.

In the case of the role pattern template the role with its domain and range must be selected from the ontology and variable names supplied. The role pattern template is displayed after the user selects the role pattern. Three buttons are displayed. The user must first select the role by browsing the ontology. OntoIQ will then lookup in the ontology the selectable domain and range associated with the role chosen by the user. Domain and range buttons can then be used to browse and select the appropriate variable, concept or instance. In the Role Pattern example the role *has\_been\_reported\_to\_have\_enzyme* with the domain *Fungi* as a variable and the range *Laccase* as an instance, is selected. In the case of the COMBO pattern, a pattern manager, the Combo Query Primitive, is provided where different combinations of nested patterns can be built from the simpler patterns. This interface links to each of the query pattern templates for the input of variables and parameters from the ontology.

After providing the ontology specific information in the pattern the user must declare completion of the user input and invoke its conversion to nRQL syntax using the ‘Confirm’ and ‘nRQL Transfer’ buttons. The pattern is translated into nRQL according to the syntax of the selected pattern. In the role pattern the user defined the role, with its domain and range, is matched to the nRQL syntax of a role pattern: Retrieve (?x) (AND (?xFungi) (?xLaccase *has\_been\_reported\_to\_have\_enzyme*))). Subsequently the user is returned to the main Query panel (Fig. 2) with the specified variable, *Fungi*, in the left variable window. Any or all variables can be selected to appear in the query results by transferring the variable to the right variable window with the direction buttons '>>'. The nRQL Syntax window then displays the corresponding syntax. For elementary queries a natural language description of the query, including the ontology specific information is provided in the Query Description panel.



**Fig. 3.** Pattern Selection panels of OntoIQ: concept query panel, role pattern query panel and a combo query panel. Screenshots shown in Fig. 2 illustrate OntoIQ query pattern functionality using the OWL file of the FungalWeb Ontology [13].

The Query button submits the syntax to the RACER server loaded with the ontology. RACER returns the query results with the namespace of the ontology elements, which can be removed by specifying the check box 'Simplify the Namespace'. Using the 'Export' button query results can be exported in XML format along with the syntax used to retrieve them.

## 2.5 Import and Export

We envision the need for researchers using the ontology for data mining of its instances to share query syntaxes. Both for the sake of speed and simplicity we have developed import and export facilities to enable the sharing of syntax between users. Fig. 4 shows the exported query syntax and query results in XML format. The <OWL> tag contains the description of the ontology to which queries were made. The <Pattern> tag describes the type of query pattern and its components as defined by the user. In the example below, a concept pattern that selects instances of the concept enzyme is shown. The corresponding nRQL syntax is described in the similarly named tag, <nRQLSyntax>. The natural language description of the query is in the <Description> tag and the instances returned by the query are in the multiple instance tags.

```
<?xml version="1.0" encoding="UTF-8" standalone="yes" ?>
- <rdf:RDF xmlns:daml="http://www.daml.org/2001/03/daml+oil#"
  xmlns:dc="http://purl.org/dc/elements/1.1/" xmlns:jms="http://jena.hpl.hp.com/2003/08/jms#"
  xmlns:owl="http://www.w3.org/2002/07/owl#" xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
  xmlns:rdfs="http://www.w3.org/2000/01/rdf-schema#" xmlns:rss="http://purl.org/rss/1.0/"
  xmlns:vcard="http://www.w3.org/2001/vcard-rdf/3.0#"
  xmlns:xsd="http://www.w3.org/2001/XMLSchema#"
- <owl:Ontology rdf:about="fungalwebontology.owl">
- <Pattern>
- <ConceptPattern>
  <Variable>a</Variable>
  <Concept>|http://a.com/ontology#Enzyme|</Concept>
  <Property>Are there instances of</Property>
</ConceptPattern>
<nRQLSyntax>( retrieve ( ?a )( ?a |http://a.com/ontology#Enzyme| ) )</nRQLSyntax>
<Description>What instances belong to the concept |http://a.com/ontology#Enzyme|</Description>
<Instances>(( ?A |http://a.com/ontology#mannan_1_4-mannobiosidase| ) )</Instances>
<Instances>(( ?A |http://a.com/ontology#alkaline_phosphatase| ) )</Instances>
<Instances>(( ?A |http://a.com/ontology#beta-glucuronidase| ) )</Instances>
<Instances>(( ?A |http://a.com/ontology#Fluorothreonine_transaldolase| ) )</Instances>
<Instances>(( ?A |http://a.com/ontology#galacturan_1_4-alpha-galacturonidase| ) )</Instances>
<Instances>(( ?A |http://a.com/ontology#Phospholipid-translocating_ATPase| ) )</Instances>
<Instances>(( ?A |http://a.com/ontology#Acetoacetyl-CoA_hydrolase| ) )</Instances>
<Instances>(( ?A |http://a.com/ontology#Adenylylsulfatase| ) )</Instances>
<Instances>(( ?A |http://a.com/ontology#Formyl-CoA_hydrolase| ) )</Instances>
```

**Fig. 4.** Export format showing the query and result generated by OntoIQ

This exported file can be uploaded to OntoIQ and re-read. The 'Import' button located on the main query panel facilitates the loading of OntoIQ XML files (Fig. 2). Fig. 5 shows the Query Sentence Import panel where exported queries

can be examined. Selecting the button marked ‘Open’ provides the user with the main query panel preloaded with the selected query for further modification or re-submission to RACER given that the corresponding ontology is loaded into OntoIQ. If the imported query does not correspond to the ontology file loaded by the user, only the query pattern is loaded without the domain specific content.

No.	nRQL Syntax	Description	Pattern S...
1	{ retrieve { ?a } { ?a [http://a.com/ontology#With_oxygen_as_acceptor]} }	What instances belonging to the...	Open
2	{ retrieve { ?vendor ?Industry } { ?vendor ?Industry [http://a.com/ontology#Producing_enzyme_for]} }	Which instances are related by l...	Open
3	{ retrieve { ?Enzyme ?Substrate ?Industry } { ?Industry ?Enzyme [http://a.com/ontology#Is_using]} { ?Substra...	What instances belonging to the...	Open
4	{ retrieve { ?Fungi } { ?Fungi [http://a.com/ontology#Laccase]} { ?Fungi [http://a.com/ontology#Has_been_reported_t...	Which instances are related to l...	Open

Fig. 5. Query Sentence Import panel for selection of available stored queries.

### 3 Conclusion

To determine the utility of an ontological conceptualization for a given application it is necessary to examine it based on its concepts, axioms and instances. This remains a time consuming challenge for domain experts who are not ready to learn new syntax in order to pose queries to ontologies or browse very large visualizations.

Despite the fact that OntoIQ requires users to learn a new approach to querying a knowledge resource it makes a significant step in reducing the technical barriers in knowledge mining. This new approach involves the following challenges: (i) how can the user identify the boundaries of the ontological conceptualization (what is included and what is not), (ii) how do users map their queries into patterns that can match the conceptualisation’s axioms in order to retrieve the desired results. The first challenge requires users to take the time browse or interrogate the ontology identifying familiar concepts and vocabulary. The second challenge can be summarized as follows; users must become familiar with the properties, the domains and ranges, used in the conceptualisation as well as the capabilities of each of the query patterns. We note domain experts have expressed concerns about the shift in thinking that requires the selection of a pattern before formulating the content of a query. This sometimes seems counterintuitive for the beginner. However we considered this challenge in the design of the tool and provide the functionality to save query patterns with a natural language description such that they can be reused at a later time by the same or less advanced users. More advanced users appear to be challenged by the need to compose new patterns to answer what may seem simple queries. Both challenges can be addressed by de-

veloping an additional tier onto the query process, namely a natural language query-answer interface which can translate natural language into query patterns. Simple axioms in the FungalWeb Ontology such as those listed below could be formulated in a number of additional ways depending on the scientific vocabulary of the user / engineer. Consequently an NLP based free-text query to query pattern mapping module would be highly relevant for providing faster and simpler access to the ontology content, for Fungal Enzyme Biotechnologists interested solely in mining the ontology content.

- *Enzyme - Has\_been\_reported\_to\_be\_found\_in - Fungus*
- *Substrate - Is\_activated\_by - Enzyme*

In summary DL-based ‘semantic mining’ is a new paradigm where OntoIQ makes the existing functionality of nRQL and RACER available to non-technical knowledge worker, albeit with a learning curve. As such it makes a relevant contribution to the evolution and uptake of knowledge discovery technology.

### Acknowledgements

This work was financed in part through the Génome Québec funded project *Ontologies, the semantic web and intelligent systems for genomics*

### References

1. Wessel M., Möller R. (2005). A High Performance Semantic Web Query Answering Engine. In Horrocks, I., Sattler, U., Wolter, F., (Eds.) *Proceedings of the 2005 International Workshop on Description Logics (DL2005)*, Whistler, BC, Canada, 2005. Available online as CEUR Workshop Proceedings Volume 147.
2. Haarslev V., Möller R. (2001). RACER System Description. R. Goré, A. Leitsch, T. Nipkow (Eds.), *Proceedings of International Joint Conference on Automated Reasoning, IJCAR. 2001*, Siena, Italy. Springer-Verlag, Berlin, pp. 701-705.
3. Protégé, <http://protege.stanford.edu/> - last accessed 2006-01-16
4. Alani, H. (2003) TGvizTab: An Ontology Visualisation Extension for Protégé. In *Proceedings of Knowledge Capture (K-Cap'03), Workshop on Visualization Information in Knowledge Engineering*, Sanibel Island, Florida, USA. ACM Press, New York, NY, USA
5. Growl, <http://www.uvm.edu/~skrivov/growl/> - last accessed 2006-01-16
6. Pellet, <http://www.mindswap.org/2003/pellet/> - last accessed 2006-01-16
7. FaCT, <http://www.cs.man.ac.uk/%7Ehorrocks/FaCT/> - last accessed 2006-01-16
8. nRQL Tab Plugin, [http://www.cs.concordia.ca/~k\\_bhoopa/nrql.html](http://www.cs.concordia.ca/~k_bhoopa/nrql.html) - last accessed 2006-01-16

- 
9. Smith B., Ceusters W., Klagges B., Köhler J., Kumar A., Lomax J., Mungall C., Neuhaus F., Rector A.L., Rosse C. (2005). Relations in biomedical ontologies. *Genome Biology*; 6(5).
  10. Birte Glimm and Ian R. Horrocks. Query answering systems in the semantic web. In Sean Bechhofer, Volker Haarslev, Carsten Lutz, Ralf Moeller (Eds) *CEUR workshop proceedings of KI-2004 Workshop on Applications of Description Logics (ADL 04)*, Ulm, Germany, Sep 24 2004. Available online as CEUR Workshop Proceedings Volume 115.
  11. Haarslev V., Moeller R., Wessel M., Querying the Semantic Web with Racer + nRQL In Sean Bechhofer, Volker Haarslev, Carsten Lutz, Ralf Moeller (Eds) *CEUR workshop proceedings of KI-2004 Workshop on Applications of Description Logics (ADL 04)*, Ulm, Germany, Sep 24 2004.
  12. The New Racer Query Language [www.cs.concordia.ca/~haarslev/racer/racer-queries.pdf](http://www.cs.concordia.ca/~haarslev/racer/racer-queries.pdf) - last accessed 2006-01-16,
  13. Shaban-Nejad A., Baker C.J.O., Haarslev V., and Butler G. (2005). The FungalWeb Ontology: Semantic Web Challenges in Bioinformatics and Genomics 4th International Semantic Web Conference (ISWC) November 6-10, 2005, Galway, Ireland. *Lecture Notes in Computer Science*, Vol. 3729, pp. 1063-1066.