Using Selectional Restrictions to Query an OWL Ontology

Leila Kosseim, Reda Siblini, Christopher J. O. Baker and Sabine Bergler
CLaC Laboratory
Department of Computer Science and Software Engineering
Concordia University
1400 de Maisonneuve Blvd. West
Montreal, Quebec, Canada H3G 1M8
{kosseim, r_sibl, baker, bergler}@cs.concordia.ca

Abstract

This paper discusses the linguistic module of an Ontology Natural Language Interaction System that is based on semantic restrictions. The system, called ONLI*, takes as input questions in unrestricted natural language, translates them into nRQL, an extension to the* RACER *ontology query language, then generates answers as retrieved by the* RACER *ontology reasoning server. Translation into nRQL is done through a syntactic analysis (with Minipar), then uses the semantic restrictions imposed by the roles stored in the ontology to map terms in the question with concepts and roles in the ontology. The system was evaluated on the FungalWeb ontology using the mean reciprocal rank (MRR) measure used in question-answering. With a test set of 36 questions, the systems achieved an MRR of 0.72*

1 Introduction

The query of knowledge representation formalisms such as ontologies is a central requirement of the Semantic Web. Increasingly we are forced to recognize the importance of providing simple query access to such knowledge repositories. Existing tools that allow users to query and reason over ontologies [1, 2, 3] use custom designed query languages [4] with complex syntax which are reportedly difficult for domain experts to master [5]. With this in mind, we sought to develop a question-answering (QA) system as a front-end to RACER, the Renamed ABox and Concept Expression Reasoner [1]. The intended users are experts who are knowledgeable in the domain, but may have little or no knowledge of the structure of the ontology. By providing a QA interface, experts can formulate their queries using natural language prose. This allows a seemingly transparent search environment, where the user does not need to formulate different syntaxes depending on the collection they are searching. Searching a document collection (such as the Web) or searching an ontology can be seen by the user as the same task.

In this paper, we present a novel approach to building a natural language front-end to an ontology that uses the semantic restrictions imposed by the ontology design to map terms in the questions to the content of the ontology. The question, formulated in unrestricted natural language, is mapped into the new RACER Query language syntax and presented to the description logic automated reasoner RACER which returns the query results.

1.1 nRQL as a Knowledge Representation Query Language

Since the recent establishment of the Ontology Web Language (OWL), design specifications for Description Logic (DL) based query languages have been proposed and existing languages contrasted, highlighting their advantages and limitations [6]. nRQL emerges as a prominent and highly expressive DL-query language and extends the existing capabilities of RACER with a series of query atoms. nRQL uses a Lisp based syntax and the general structure of a query is composed of a query head e.g. `retrieve(?x)` upon which variables used in the body are projected e.g. `(?x Fungi)`, where `(retrieve (?x)(?x Fungi))` queries, for instance, for the concept `Fungi`. In this paper, we employ conjunctive queries where the atoms are simple concept or role assertions and where the variables in the body of the query match the corresponding individuals in the ontology that satisfy all query conditions. A detailed description of nRQL is given in [7] and verbose examples are outlined in [8].

1.2 The FungalWeb Ontology

The NL interface was developed and tested on the FungalWeb Ontology [9]. The FungalWeb Ontology is a prototype bio-ontology, scripted in the OWL formalism. It is an integrated conceptualization of multiple scientific domains. These overlapping domains include taxonomies of fungi and enzyme reaction mechanisms as well enzyme substrates and industrial specifications describing the applications and benefits of enzymes. The FungalWeb Ontology is a large scale ontology comprising 3616 concepts and 11,163 instances related by 142 roles. The conceptualization was designed so that fungal species, enzyme names, enzyme product names, enzyme vendor names an chemical names are modeled as instances. Free text segments describing enzyme applications, industrial benefits of enzymes were also modeled as instances. The following example illustrates the query capability of the conceptualization e.g. if the user is looking for vendors selling enzyme products that contain Xylanase, the user composes the nRQL syntax below:

```
(retrieve (?x) (AND (AND
  (?x ?y <http://a.com/ontology#Sells>)
  (?y ?z <http://a.com/ontology#Contains>))
  (?z <http://a.com/ontology#Xylanase>)))
```

The scope of the ontology has been further illustrated in a series of application scenarios [10, 11] demonstrating the range of query capabilities afforded by the conceptualization.

During the development of the FungalWeb Ontology there was a need for domain experts to interact directly with the conceptualisation. The Ontoligent Interactive Query Tool (OntoIQ) [12] was subsequently developed to provide browse and click query functionality to the ontology using nRQL. OntoIQ is useful in the context of ontology development and has been able to identify challenges in the ontology query paradigm that could be overcome by Natural Language Query access to nRQL.

2 Previous Work on querying ontologies

Natural language interfaces to databases have been a significant research focus, especially during the 70s and 80s (e.g, [13, 14, 15]). However, only recently has the topic of natural language interfaces to ontologies been seriously investigated. Earlier work has been on restricted language or simplified English (e.g. [16]) where constraints are imposed on the expressiveness of the user's prose. For example, [17, 18] use the Attempto Controlled English (ACE) to query the semantic Web. The interface imposes some structure on the user's input to guide the entry but does not restrict the user with an excessively formalistic language. Each ACE query is translated into a discourse representation structure that is then translated into an N3-based semantic web querying language (RDQL), which allows their execution. Their work shows that these kinds of interfaces are simple to use and provide superior retrieval performance to traditional logic-based approaches when used by a casual user. However, to provide an easier to use interface, unrestricted language seems more *natural*, especially for the occasional user.

Few systems however allow questions in unrestricted English. AquaLog [19] and its successor PowerAqua [20], for example, are ontology-driven QA systems, which take an ontology and a natural language question as an input and return answers drawn from semantic data compliant with the input ontology. AquaLog was tested on the KMi ontology on academic life. Its linguistic module makes use of the Gate package [21] and uses several metrics to compute the similarity between terms in the question and terms in the ontology based on string-based algorithms and WordNet. This paper presents a system similar to AquaLog and PowerAqua, but relies on semantic restrictions to improve its linguistic module.

3 The ONLI Natural Language Interface

3.1 Overview

The system consists of 3 main modules in addition to a web user interface that allows the user to interact with the system. The question is first processed by the Minipar [22] general purpose syntactic parser. The resulting dependency parse tree is then analysed to extract all predicate-argument structures that will be mapped to the concepts and roles of the selected ontology. The mapping is done using the selectional restrictions imposed by the ontology. The result is a set of mappings along with their confidence score. The next module of the system then generates nRQL queries using hand-made transformation rules, and executes the highest scoring query over the ontology. Let us now describe each module in detail.

3.2 Syntactic Analysis

Initially, the user's question is parsed to identify its syntactic constituents. For this, we use the general purpose an freely available Minipar parser [22].

Predicates and noun phrases contain the main semantic information of the question. These will thus be extracted from the question and mapped to roles, concepts or instances in the ontology. Roles in the FungalWeb ontology can be binary, relating two concepts to each other; or unary, relating one instance to a concept. Therefore, from the parse

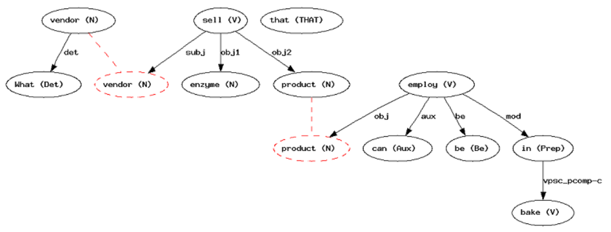

Figure 1. MINIPAR parse tree for the sentence *What vendors sell enzyme products that can be employed in baking bread?*

tree of a question, all sets of predicates along their arguments are extracted and represented into a predicate structure made of the triplet argument-predicate-argument.

In the parse tree, these predicate structures may be realized by many grammatical relations: a verb with its subject (deep or surface) and object (direct or not), a adjective modifying a noun, a noun-noun compound, a passive verb modifying a noun ... or may need to be extracted from more complex verb phrases (e.g. an inverted auxiliary *is it sold?*). To be more concrete, consider the question:

What vendors sell enzyme products that can be employed in baking bread?

MINIPAR will generate the dependency tree shown in Figure 1. From this structure, two predicate structures must be extracted: the predicate *sell* and its arguments *enzyme product* and *vendors*, and the predicate *employ* and its arguments *enzyme products* and *baking bread*. These 2 predicate structures are shown below:

`arg1`	`pred`	`arg2`
vendor	sell	enzyme product
enzyme product	employ	baking bread

Although not shown in Figure 1, in the parse tree, the predicate *sell* introduces an `i` relation (a relation between the main clause and the inflectional phrase). Similarly to the predicate *employ* that also introduces an `i` relation. Arguments can participate in different types of grammatical relations. *vendor* is a subject, the argument *enzyme* is an `obj1`, while *product* is an `obj2` and *bread* is an `obj`.

Given this variety of syntactic realisations, in order to extract the predicate structures for each question, 19 handcrafted rules were developed. These rules take into account the grammatical relation and the syntactic category of each word as extracted from the Minipar tree. For example:

- If a noun is being modified by a relative clause, then this relative clause is treated as a new predicate structure with the noun as one of its arguments.
- If a word is an adjective or noun it can be treated as a predicate if it participates in a grammatical relation `Pred` related to a *to be* clause. In this case any noun-noun modifier (`nn`) is treated as an argument to this predicate and any subjects to verb *to be* is treated as the other argument.

To develop these rules, we analysed the output of Minipar on a set of 180 training questions (see section 4). The rules are applied recursively to embedded structures to find all predicate structures from the question.

The argument slot of a predicate structure may be left empty if the argument was elided in the question or if a relative determiner was used and Minipar could not identify its antecedent. An empty argument slot will be replaced by a variable name in the final nRQL query (see section 3.4).

The predicate slot may also be empty. Such empty predicates occur if the predicate is in the first position in the question (e.g. *Find enzymes, Give enzymes.*) or is a verb *to-be*. Such predicates are treated differently, because they will map to a unary concept rather than to a binary role.

Table 1 shows other examples of questions along with their predicate structures.

Question	**Predicate Structure**
What enzyme can be used in baking bread?	`(arg1:enzyme, pred:`*`use`*`, arg2:`*`baking bread`*`)`
Find an enzyme that can be used in baking bread.	`(arg1:∅, pred:∅, arg2:`*`enzyme`*`)` `(arg1:∅, pred:`*`use`*`, arg2:`*`baking bread`*`)`
Is protease an enzyme?	`(arg1:`*`protease`*`, pred:∅, arg2:`*`enzyme`*`)`
What is used in baking bread?	`(arg1:∅, pred:`*`use`*`, arg2:`*`baking bread`*`)`

Table 1. Examples of predicate structures

3.3 Ontology Mapping

Once all predicate structures have been extracted from the question, we attempt to match each constituent of the structure to variables, concepts, instances or roles in the ontology. This can be seen as a classical categorization problem, in particular, Word Sense Disambiguation. Indeed, the task here is to find a function to map the linguistic expressions to particular senses (concepts, instances or roles in the ontology).

Domain Concept	Role	Range Concept
substrate	is activated by enzyme	enzyme
enzyme	can be used	commercial enzyme product
fungi	grows on substrate	substrate
industrial and environmental process	is using	commercial enzyme product
industrial and environmental process	is using	fungi
industrial and environmental process	is using	enzyme
...	...	...

Table 2. Examples of roles and their domain and range in the FungalWeb ontology

To select the correct mapping we were inspired by the selectional restriction-based disambiguation approach used in word-sense disambiguation [23]. Indeed, in a text, a predicate often imposes semantic constraints on its arguments which allow to disambiguate its sense, and in turn, the sense of its arguments. For example, in its transitive form, the verb *drink* imposes that its direct object be a *liquid*. The correct sense of an ambiguous direct object can therefore be identified through this semantic constraint. With an ontology, this same strategy can be used as the roles in the ontology impose constraints on the domain and range of the concepts they can relate. In turn, correctly identifying the concepts or instances involved in the question can help us identify an ambiguous role.

For example, Table 2 shows selected roles in the FungalWeb ontology along with their corresponding domain and range concepts. As the table shows, a concept for an argument may not be semantically compatible with all roles. For example, if the predicate has been mapped to the role `grows on substrate`, then the arguments must map to either the concepts (or instances of) `fungi` or `substrate`.

For each predicate structure, the result of the semantic analysis is a list of possible roles, concepts, and instances in the ontology along with a confidence measure. For efficiency reasons, predicates are only mapped to roles, while the arguments are mapped to concepts and instances. Empty arguments in the predicate structure are mapped to variables.

3.3.1 Mapping predicates

Because predicates do not exhibit much domain terminology, we can use general purpose lexical resources to match them to roles in the ontology. To map a predicate of the question to a role in the ontology, we try to unify its stem (identified by Minipar) with a role of the ontology. For this, we use a measure of semantic similarity.

To compute the semantic distance between a predicate and a candidate role, we use WordNet::Similarity [24, 25]. This package uses WordNet and, by default, the path length between 2 words to compute their semantic relatedness. The predicate in the sentence should be a correct English term, however, the name of the role in the ontology is not restricted to be available in WordNet. In fact, many role names in the FungalWeb ontology do not correspond to English verbs, they are often multi-word terms (e.g. `is activated by enzyme`). In the case of multi-word terms, we add the appropriate words to the role to form a complete sentence If a role name is given, then we parse the sentence and use the head of the sentence to be compared using WordNet similarity to the retrieved predicate.

If the semantic similarity returns zero, then a default small value (ϵ) of 0.001 is used, so as not to rule out any mapping.

3.3.2 Mapping arguments

Once the predicate is matched to a set of possible roles, we try to match its arguments to a variable, or a concept or an instance in the domain and range of this role. This is where semantic restrictions come into play.

If the argument is empty, then a new variable is created as a placeholder, but the domain and range of the role already mapped are kept as constraints to the variable.

If the argument is not empty, then we need to map it to a concept or an actual instance. Because NPs tend to be much more domain specific (e.g. enzyme names, substrate names, ...), general purpose lexical resources such as WordNet cannot be used to match NPs. Instead, we use a measure of lexical similarity.

To score the lexical relatedness between an argument and a concept or an instance, we use the inverse of the edit distance – the minimal number of characters needed to make the 2 strings identical. The lexical score is also normalised to be within 0 and 1.

If the argument matches a concept and an instance we give the matched instance a slightly higher score over the matched concept. If the lexical similarity returns zero, then we use the same ϵ value.

The Cartesian product of all possible mappings for the predicate and all possible mappings for the arguments is then computed. The overall score of the final mapping is computed as the product of the individual mappings. Table 3 shows an example.

constituent	mapping	score
pred: *sell*	role: `sells`	1
arg1: *vendor*	concept: `vendor name`	× 0.498
arg2: *enzyme product*	concept: `commercial enzyme product`	× 0.66
		0.328
pred: *sell*	role: `producing enzyme for`	0.333
arg1: *vendor*	concept: `vendor name`	× 0.498
arg2: *enzyme product*	concept: `industrial processes`	× 0.01
		0.001

Table 3. Examples of semantic mapping for the predicate structure `arg1:vendor -- pred:sell -- arg2:enzyme product`

3.3.3 Variables Co-referencing

Once a set of possible mapping is built for each predicate structure of the question, we need to make sure that variables that should refer to the same entities actually do. This, in effect, allows us to process the predicate structures of a question as a single semantic unit, rather than a conjunction of unrelated predicate structures.

To identify which variables should co-refer to the same entities, we use the semantic constraints we set when we used the semantic relations to bound our variables (see section 3.3.2). If the constraints of two variables can be unified, then we consider the variables to co-refer. Since we already know the possible concepts that this variable could belong to, we gave the variable name the concept id, so referring to the same variable and thus the same id leads us to create the relationship between the predicate structure. This strategy seems to work with the current size of the FungalWeb ontology, but may not scale up to a larger ontology. In this case, a deeper analysis of the question is probably needed.

For each question, the list of the possible mappings is finally ranked according to the overall confidence score and the best 10[1] are sent to be translated to nRQL.

3.4 Querying with a Reasoner

The reasoner query module is responsible for creating the nRQL query and sending them to the RACER reasoner. In the current state of the project, only two types of nRQL queries have been considered: unary concept queries and binary role queries.

[1]This threshold was set arbitrarily.

concept1 role concept2 `(?x <concept1>) (?y <concept2>) (?y ?x <role>)`
variable1 role variable2 `(retrieve (?y ?x) (?y ?x <Role>))`
variable ⊘ concept `(retrieve (?x) (?x <concept>))`

Table 4. Examples of predicate structure patterns and corresponding nRQL queries

A unary concept query tries to find instances of a particular concept (e.g. *Find all enzymes* ⇒ `(retrieve (?x) (?x enzyme))`) or to determine if an entity is an instance of a concept (e.g. *Is Protease an Enzyme?* ⇒ `(retrieve () (Protease Enzyme))`). A unary concept query can therefore have one of the two forms:

1. `(retrieve (?x) (?x <concept>))`
2. `(retrieve () (<instance> <concept>))`

A binary role query searches for the binding between 2 concepts or instances (e.g. *What can be used in what?* ⇒ `(retrieve (?y ?x) (?y ?x can be used in))`). A binary role can specify particular concepts or instances instead of specifying a variable. For example, *What can be used in baking?* ⇒ `(retrieve (?x) (?x <Baking> can be used in))`, or *Can Protease be used in baking?* ⇒ `(retrieve () (<Protease> <Baking> can be used in))`. A binary role query can therefore take the 4 following forms:

3. `(retrieve (?y ?x)(?y ?x <role>))`
4. `(retrieve (?x)(?x <instance> <role>))`
5. `(retrieve (?x)(<instance> ?x <role>))`
6. `(retrieve ()(<instance> <instance> <role>))`

In order to create the nRQL queries from the mappings, we created cases for all possible combinations of the argument-predicate-argument triplet. An argument could have a variable, an instance, or a concept and the predicate could be a empty or not. That leads us to 18 different combinations, and for each combination, a hand made rule is created to produce one or more appropriate nRQL statements. Table 4 shows a sample of these patterns and the corresponding nRQL expression.

For example, from the question in section 3.2 *What vendors sell enzyme products that can be employed in baking bread?*, 2 predicate structures were extracted:

```
arg1:vendor
pred:sell
arg2:enzyme product
```

and

```
arg1:vendor
pred:sell
arg2:enzyme product
```

The corresponding highest ranking ontological matches are:

```
concept:vendor name
role:sells
concept:commercial enzyme product
```

and

```
concept:commercial enzyme product
role:can be used in
instance:baking
```

The first triplet corresponds to the case of `concept1 role concept2` shown in Table 4 which creates 3 nRQL statements:

1. `(?x <vendor name>))`
2. `(?y <commercial enzyme product>))`
3. `(?y ?x <sells>)`

The second triplet corresponds to the case of `concept role instance` which creates 2 nRQL statements:

1. `(?y <commercial enzyme product>))`
2. `(?y <baking> <can be used in>))`

Notice how the two triplets are related using the variable `y`, which co-refer to the same concept *commercial enzyme product*). Individual nRQL expressions are then connected with an AND operator.

4 Evaluation

To develop and test the system, we used a corpus of 206 pairs of questions and their associated nRQL queries. The material was created by 4 different casual users in order not to be influenced by the writing style of one particular person. The users were all knowledgeable in the domain and the content of the ontology, but did not necessarily know its structure and role names.

¿From the original 206 question-query pairs, 26 were discarded because they involved issues we do not consider (e.g. quantifiers, see section 5). From the 180 remaining, 80% randomly selected pairs were used to develop the system (develop the syntactic rules, ...) and the 20% remaining were used for the evaluation.

We thus evaluated the prototype using 36 questions and compared the system-generated results with the corresponding queries as gold-standard. The comparison was based on query equivalence. If the generated query was not equivalent to the one in the gold-standard, it was considered wrong. For each question, the system generates a set of possible queries ranked in order for confidence. For each question *q*, we therefore computed the final score as the reciprocal rank of their first correct answer. If none of the generated queries was equivalent to the gold-standard, a score of 0 was given. Otherwise, the score is equal to the reciprocal of its rank. For example, if a question generated 4 ranked queries, and the 3^{rd} one is correct, the question received a score of $\frac{1}{3}$. The overall system score is the average $RR(q)$ for all questions q. This methodology is called the mean-reciprocal rank (MRR) score as used in question-answering [26].

For all 36 questions, the MRR was 0.72. Out of the 36 questions, 24 were found at rank 1; 6 questions ranked between 2 and 10, and 6 were not found in the top 10 answers. Out of the 6 questions that were not translated properly, 3 were not parsed correctly by Minipar, 2 did not match correctly the actual instance in the ontology, and 1 contained a hidden relation.

Hidden relationships involve deeper reasoning, For example, in *Which proteins are known to act on p-aminophenol?* the relation `act on` relates a substrate to an enzyme, but the question is looking for a protein, so we need to recognise that an enzyme is a protein, which means we must also check the subsumption hierarchy of the ontology for *is a* relations. Hidden relationships were not considered in the system analysis.

5 Conclusion and Future Work

In this paper we have described a system that acts as natural language interface system for a domain ontology. The interface takes as input questions in unrestricted language and translates them into nRQL queries that are then run over the RACER reasoner. The linguistic analysis of the questions is based on a syntactic parse from which a set of predicate structures is extracted, which are mapped to the correct ontology entities using semantic restrictions imposed by the ontology structure. We tested the interface on the FungalWeb ontology, with real questions composed by four different casual users, and obtained an MRR of 0.72.

Further work includes a more robust evaluation of the system with larger, more diverse ontologies and a larger test collection. We presume that the semantic restrictions approach worked well in our case because each role was not used in many contexts, but as roles can relate a larger number of domains and ranges, the approach may not scale up if the semantic and lexical mappings do not hold.

In addition, several linguistic phenomena are not taken into account. For example, conjunctions and disjunctions within noun phrases, as well as quantifiers (*some, three, . . .*) are simply ignored. For example, *Find three fungi that have been reported to have Pectinase* will be considered equivalent to *Find all fungi that have been reported to have Pectinase*. In the case of conjunctions or disjunctions (e.g *What fungi have been reported to have Pectinase and/or Cellulase?*), the system will not be able to recognize and deal appropriately with the issue. In the case of quantifiers, the system will ignore the quantifier and will search for all possible answers.

Another drawback is our limitation to use noun phrases that are lexically close to the vocabulary used in the ontology for concepts and instances. Mapping of noun phrases can only be performed at the lexical level because a general purpose semantic lexicon for general English is not helpful for domain specific terminology. In the biology domain, terminology plays a central role, and several domain-dependant synonyms are typically used to refer, for example, to enzymes or proteins. Without a dictionary of the domain, only lexical matches can be made. In our experiment, this did not cost a lower performance, but if we use a larger ontology or a larger pool of users, the lexical similarity alone may not scale up.

Acknowledgments

This project was financially supported by Genome Québec. Many thanks to Mike Carrick for this help in building the question set.

References

[1] Haarslev, V., Möller, R.: RACER System Description. In Goré, R., Leitsch, A., Nipkow, T., eds.: Proceedings of International Joint Conference on Automated Reasoning (IJCAR-2001), Sienna, Italy, Springer-Verlag, Berlin (2001) 701–705

[2] Pellet: http://www.mindswap.org/2003/pellet/. (last accessed 2006-01-16)

[3] FaCT: http://www.cs.man.ac.uk/ horrocks/fact/. (last accessed 2006-01-16)

[4] Wessel, M., Molle, R.: A high performance semantic web query answering engine. In: Proceedings of the 2005 International Workshop on Description Logics (DL2005), Whistler, Canada (2005)

[5] Smith, B., Ceusters, W., Klagges, B., Kohler, J., Kumar, A., Lomax, J., Mungall, C., Neuhaus, F., Rector, A., Rosse, C.: Relations in biomedical ontologies. Genome Biology **6** (2005)

[6] Glimm, B., Horrocks, I.R.: Query answering systems in the semantic web. In Bechhofer, S., Haarslev, V., Lutz, C., Moeller, R., eds.: CEUR Workshop Proceedings of KI-2004 Workshop on Applications of Description Logics (ADL 04), Ulm, Germany (2004)

[7] Haarslev, V., Moeller, R., Wessel, M.: Querying the semantic web with racer + nrql. In Bechhofer, S., Haarslev, V., Lutz, C., Moeller, R., eds.: CEUR Workshop Proceedings of KI-2004 Workshop on Applications of Description Logics (ADL 04), Ulm, Germany (2004)

[8] nRQL: The new racer query language. (www.cs.concordia.ca/haarslev/racer/racer-queries.pdf)

[9] Shaban-Nejad, A., Baker, C.J., Butler, G., Haarslev, V.: The FungalWeb Ontology: Semantic Web Challenges in Bioinformatics and Genomics. In: 4th International Semantic Web Conference (ISWC). Lecture Notes in Computer Science 3729, Galway, Ireland (2004) 1063–1066

[10] Baker, C., Witte, R., Shaban-Nejad, A., Butler, G., Haarslev, V.: The fungalweb ontology: Application scenarios. In: Eighth Annual Bio-Ontologies Meeting, co-located with ISMB 2005, Detroit, Michigan (2005)

[11] Baker, C., Shaban-Nejad, A., Xu, S., Haarslev, V., Butler, G.: Semantic web infrastructure for fungal enzyme biotechnologists. Journal of Web Semantics: Special Edition on Semantic Web for the Life Sciences (2006) in press.

[12] Baker, C., Xu, S., Butler, G., Haarslev, V.: Ontoligent interactive query tool. In: Proceedings of Canadian Semantic Web Working Symposium. (2006) in press.

[13] Androutsopoulos, I., Ritchie, G., Thanisch, P.: Natural language interfaces to databases–an introduction. Journal of Language Engineering **1** (1995) 29–81

[14] Copestake, A., Sparck-Jones, K.: Natural language interfaces to databases. Knowledge engineering Review **5** (1990) 225–249

[15] Clifford, J.: Natural language querying of historical databases. Computational Linguistics **14** (1988) 10–34

[16] Decker, S., Erdmann, M., Fensel, D., Studer, R.: ONTOBROKER: Ontology Based Access to Distributed

and Semi-Structured Information. In Meersman, R., ed.: Database Semantics: Semantic Issues in Multimedia Systems, Proceedings TC2/WG 2.6 8th Working Conference on Database Semantics (DS-8), Rotorua, New Zealand, Kluwer Academic Publishers (1999) 351–369

[17] Bernstein, A., Kaufmann, E., Fuchs, N.E.: Talking to the Semantic Web - A Controlled English Query Interface for Ontologies. AIS SIGSEMIS Bulletin **2** (2005) 42–47

[18] Bernstein, A., Kaufmann, E., Gohring, A., Kiefer, C.: Querying ontologies: A controlled english interface for end-users. In: Proceedings of the 4th International Semantic Web Conference. (2005) 112–126

[19] Lopez, V., Pasin, M., Motta, E.: Aqualog: An ontology-portable question answering system for the semantic web. In: Proceedings European Semantic Web Conference (ESWC), Crete (2005) 546–562

[20] Lopez, V., Motta, E., Uren, V.: Poweraqua: Fishing the semantic web. In: Proceedings European Semantic Web Conference (ESWC), Montenegro (2006)

[21] Cunningham, H.: GATE, a General Architecture for Text Engineering. Computers and the Humanities **36** (2002) 223–254

[22] Lin, D.: Dependency-based evaluation of MINIPAR. In: Proceedings of the Workshop on the Evaluation of Parsing Systems, Granada, Spain (1998)

[23] Resnik, P., Yarowsky, D.: A perspective on word sense disambiguation methods and their evaluation (1997)

[24] Pedersen, T., Patwardhan, S., Michelizzi, J.: Wordnet::similarity - measuring the relatedness of concepts. In: Proceedings of the Nineteenth National Conference on Artificial Intelligence (AAAI-04), San Jose, California (2004)

[25] Patwardhan, S., Banerjee, S., Pedersen, T.: Using measures of semantic relatedness for word sense disambiguation. In: Proceedings of the Fourth International Conference on Intelligent Text Processing and Computational Linguistics (CICLING-2003). (2003)

[26] Voorhees, E.: Overview of the TREC 2001 Question Answering Track. In: Proceedings of The Tenth Text REtrieval Conference (TREC-X), Gaithersburg, Maryland (2001) 157–165