# Metafrastes: A News Ontology-Based Information Querying Using Natural Language Processing

Hanno Embregts, Viorel Milea, and Flavius Frasincar

Econometric Institute, Erasmus School of Economics, Erasmus University Rotterdam
P.O. Box 1738, 3000 DR Rotterdam, The Netherlands
hanno.embregts@gmail.com, milea@ese.eur.nl, frasincar@ese.eur.nl

**Abstract.** This paper presents Metafrastes, a system that provides users with the ability to retrieve information from Semantic Web knowledge bases through queries that are formulated in natural language. The Web system that we introduce is engineered based on several Semantic Web tools and techniques. Our contribution consists mainly of a Natural Language Processing engine, able to translate queries formulated in natural language to SPARQL queries that can be applied to existing knowledge bases. Additionally, we develop a user interface that captures the user interaction with the system. Last, we evaluate the system based on a number of pre-defined queries formulated in natural language. The proposed approach has been positively evaluated with respect to precision and for queries that are aware of the information structure from the knowledge base.

## 1 Introduction

Since the introduction of the World Wide Web, the amount of information that is available via this medium has steadily increased [3]. Nowadays, millions of Web pages provide enormous amounts of information to anyone with access to it. When dealing with such massive amounts of information, it is very important that the information is organised in a structured way in order to optimally serve the Web user's information needs. The Semantic Web [5] helps in addressing this issue. Designed as an extension of the current World Wide Web, the Semantic Web strives to assign proper meaning to the information, with the ultimate goal to present the information in such a way that both human users and machines can use and manipulate the information to better suit their wishes. To support this vision, numerous technologies have been developed, such as data representation languages RDF [6,13], OWL [16], and tOWL [17,18], and query language SPARQL [19]. RDF and OWL enable the representation of self-describing data, on which queries can be executed using SPARQL. The more recent tOWL language extends OWL with a temporal dimension, enabling representations of change and state transitions, in addition to static information.

In this paper we focus on user interaction with Semantic Web information systems. The Semantic Web brings numerous useful additions to the current

World Wide Web, but these additions will only be of use if they enable human users or automated processes to interact with Web information systems. Various types of user interaction exist, including search term narrowing (which enables the user to start with a broad term, while narrowing it down to more specific terms by, e.g., expanding a taxonomy tree filled with search terms) and natural language input (which enables the user to start his search for information by formulating a question in natural language). Natural language input can be a convenient way for the user to retrieve information, because there is no need to summarise his thoughts in a single keyword; the user can merely type his information need in natural language in order to retrieve the desired information.

To clarify the usefulness of natural language input, we present an example. Suppose that a news ontology exists, that automatically collects news items about, e.g., companies. A user might then be interested in the news items on the competitors of Google. However, without knowing the structure of the underlying ontology, this user cannot know which companies are competing with Google. Being able to formulate a natural language query such as "In which news items does at least one of Google's competitors appear?" would be a great feature for this user, as this user does not need to know exactly which companies compete with Google, yet he can still obtain relevant answers.

To be able to use natural language input, a natural language processing engine needs to be in place to convert the user's question to a format that can be processed by a computer. This is why we focus on user interaction by using and analysing some existing natural language processing engines, both designed for conventional use and for use in Semantic Web environments. Based on this analysis, a Semantic Web approach to natural language processing of user input is proposed.

The outline of this paper is as follows. In Section 2 we provide an overview of related work. In Section 3 we introduce the NLP engine that we develop for querying knowledge bases based on queries formulated in natural language. An evaluation of the system is presented in Section 4. Finally, we conclude in Section 5.

## 2 Related Work

Several systems have already been presented in literature that deal with queries based on natural language. In this Section we discuss CO-OP, Masque/SQL, AquaLog, and SemNews, as prolific examples of such systems.

Cooperative Query System (CO-OP) [11] is a portable natural language database query system which was developed in the mid-1980s and meant to work with a CODASYL database management system. It focuses on providing solutions for a number of long-recognised NLP issues, including ambiguity and vagueness. Additionally, it also aims to reduce the installation effort that is required whenever an NLP system is applied to a different domain.

Masque/SQL [4] is a portable natural language front-end, which answers English questions by generating and executing SQL code. It is a modification of

the original MASQUE (Modular Answering System for Queries in English) system, which was focused on working with Prolog queries on Prolog databases. The MASQUE system, which was developed at the Artificial Intelligence Applications Institute and the Department of Artificial Intelligence of the University of Edinburgh, strives to combine extensive linguistic coverage, efficiency and portability. Moreover, it was also developed to be easily configurable for different knowledge domains.

AquaLog [14] is a question-answering tool that takes an ontology and a natural language query as input, after which it returns an answer from the ontology. The natural language queries are translated into logical queries, which are interpreted with respect to a given ontology and the corresponding semantic markup. Being an ontology-compatible system, AquaLog strives to work with Semantic Web technologies only, thereby trying to be as portable as possible by making use of the ontology standard representation languages.

SemNews [9,10] is a semantic news service that monitors different RSS news feeds and provides structured representations of the meaning of the news items that originate from these news feeds. It strives to make more Semantic Web content on the Web available by extracting summaries from RSS descriptions of news items and by processing this natural language input using an underlying component called OntoSem.

In deciding the usefulness of previous approaches for our current endeavour, we define four criteria that these approaches should fulfil: i) Semantic Web foundation or orientation, ii) ability to answer natural language questions, iii) ability to deal with news items, iv) use of an intermediate representation.

Table 1 provides an overview of the related work that was discussed in this section, along with the four criteria that we deem relevant. From this table, it follows quickly that AquaLog is the most suitable NLP system to consider. We apply AquaLog on our domain ontology (a news ontology) and the results were not adequate. In fact, none of the four complete NLP systems were able to address our needs, therefore we chose not to use an existing NLP system, but a partial one in the form of a linguistic parser. Using such a parser as a first step in our NLP engine provides us with flexibility, as the resulting parse tree can be altered whenever we deem necessary.

## 3 NLP Engine

The architecture of the NLP engine that we introduce is shown in Figure 1. We use the remainder of this section to discuss the design of the architecture.

To start the process, the user formulates an English question and commands the NLP engine to process his question. The question is then parsed by a *parser*,

[1] The 'partial' score was assigned because the underlying data structures of this related approach resemble the structure of ontologies.

[2] The 'partial' score was assigned because the used intermediate representation did not result in adequate portability, as some components of the system would have to be heavily altered to apply them to a new domain.

**Table 1.** Overview of the related work and their scores on the four criteria

| Criterium | CO-OP | Masque-SQL | AquaLog | SemNews |
|---|---|---|---|---|
| **Semantic Web** | partial[1] | partial[1] | yes | yes |
| **NL questions** | yes | yes | yes | no |
| **News items** | no | no | no | yes |
| **Intermediate** | partial[2] | yes | yes | partial[2] |

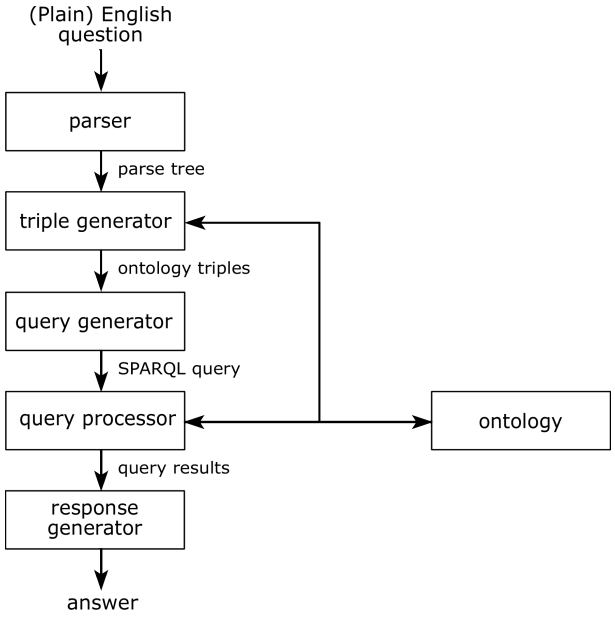


**Fig. 1.** The architecture of our NLP engine

which returns a parse tree. The parse tree is used as input for the *triple generator*, which then generates ontology triples based on its input. The ontology triples are then forwarded to the *query generator*, which constructs the correct SPARQL query. The SPARQL query is processed by the *query processor*, which interacts with the ontology to obtain the query results. An appropriate answer to the user question is generated by the *response generator*, which takes the raw query results and reformats them in a user-friendly format, as shown in Figure 6.

Our architecture uses a sequential structure, an approach to systems design that is quite common in the field of natural language processing. This is because a sequential structure provides flexibility when it comes to the format of the final results, as the engine can be easily altered to work with a different query language, e.g., SPARQL 1.1 [8], by just altering two components of the total five that are involved in the whole process (for the RDQL example, only the query generator and the query processor would need to be altered). Apart from the

sequential structure, the intermediate representations (the parse tree and the ontology triples) provide more possibilities to generate multiple types of output.

### 3.1 Implementation

This section describes the decisions that were made in order to come to a valid implementation. After describing shortly how the implementation was organised and set up, we discuss the implementation by focusing on every separate component of the engine, as displayed in Figure 1.

**Organisation and Setup** Our NLP engine is implemented as a Java program. We call the system 'METAFRASTES' (original Greek word: $\mu\epsilon\tau\alpha\varphi\rho\alpha\sigma\tau\eta\varsigma$), literally meaning 'translator' in Greek.

*Tools and Languages* To be able to implement the engine, a number of different tools and languages are used: i) The Java Platform (Standard Edition 6 Release), ii) Resource Description Framework for the Semantic Web (RDF), iii) the Web Ontology Language (OWL), iv) SPARQL (Query Language for RDF), v) Jena (Semantic Web Framework for Java) [15], vi) ARQ (SPARQL Processor for Jena) [1], vii) Hermes (Semantic News Portal) [2], and viii) Stanford Parser (Statistical Parser) [12].

**Parsing** With most NLP engines, *parsing* is the first phase in the process of translating the user input. It is used to interpret the user input syntactically, so that the structure of the user input is made available to the NLP engine. The process of parsing results in a parse tree, which represents the user input in such a way that it can be handled by the NLP engine.

*Stanford Parser* The tool that we use for the parsing step is the Stanford Parser [12]. It consists of a Java implementation of probabilistic natural language parsers and can be used to produce the *most likely* analysis of a user inputted sentence. An example of a parse tree as it can be produced by the Stanford Parser is shown in Figure 2. The example question we use is: "Which company has Steve Ballmer for CEO?". This example will be used throughout the remainder of this paper to illustrate the workings of different system components.

The parse tree in Figure 2 was generated by inputting the example user question in the lexicalized Probabilisty Context-Free Grammar (PCFG) parser, which uses various forms of probabilistics and statistics to annotate important words in the user input as phrasal nodes. These phrasal nodes are all assigned an annotation, specifying the type of the word or part of sentence.

**Domain Ontology** The domain ontology was built by means of a wrapper application on Yahoo! Finance to extract information on the top-100 NASDAQ

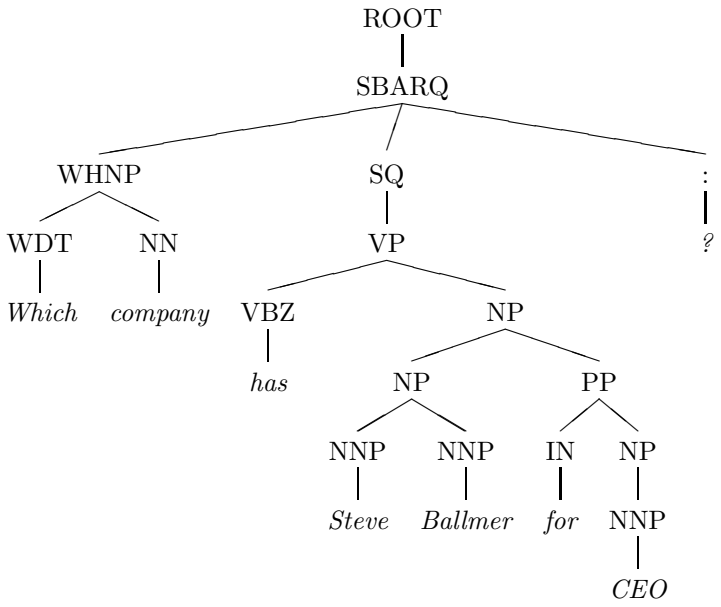


**Fig. 2.** "Which company has Steve Ballmer for CEO?": parse tree

companies. The OntoClean approach [7] was used to validate the domain ontology composed of entities as companies, products, persons, currencies, CEOs, etc. Each concept has a number of lexical representations associated (some derived from the corresponding WordNet entries, where these are available).

**Generating Triples** The most important step in the process is the generation of ontology triples. These triples are of the form `(subject predicate object)`. In this section, we zoom in on how the ontology triples are generated and what is required in order to carry out this process. We start by first introducing the Jena tool in more detail. Next, we describe the heuristics that we developed for Metafrastes.

*Jena* Jena is a Semantic Web framework for Java. This tool provides functionality for communication with RDF data sets and ontologies. We use it in our NLP engine to query the ontology. Furthermore, Jena contains a separate query processor called ARQ, which is able to process SPARQL queries and return the results from the ontology. We use this part of Jena later on in the question translation process.

*Developing Heuristics* When a complex problem needs to be solved, it may prove very helpful to develop heuristics that describe how a solution to the problem may be obtained.Such heuristics aim to exploit the problem specifics and to

produce a suboptimal solution for a complex problem. The heuristics developed for our system are shown in the algorithm depicted in Figure 3.

The heuristics take the parse tree as input, which was the result of the previous step in the question translation process. Then, on lines 1 and 2, two variables are assigned. The variable *currentTerm* keeps track of the current term in the parse tree, while *keywords* is used when a request for news items needs to be discovered.

From line 3, we check whether the user asked for news items in his question. If so, the value of *currentTerm* is updated with the news item term. The *if* tag on line 6 determines whether there are relations linked to the news item term. If so, the *currentValue* variable is updated again and ontology triples are added for this relation. This step repeats itself, but now a link between the current term and an ontology concept is searched for. If this is found, the ontology triples are again added and next to updating *currentTerm*, the previous term is stored. After that, we check whether there are any links to ontology concepts in the remaining leaves of the parse tree. If this is so, ontology triples are added for every link to an ontology concept that exists. If no links are present, a final triple is added (the synonym triple), after which the ontology triples are returned.

If the user did not ask for news items, the algorithm skips to line 22, where we check whether the parse tree contains an ontology concept. If this is the case, an initial ontology triple on the concerned ontology concept is added, after which the while-loop is entered to search for more linked concepts. Again, they are added as ontology triples until no more linked concepts exist. The assembled ontology triples are returned at the end. Figure 4 shows the ontology triples for our example after they have been returned by the ontology triple generator.

**Generating Query** The SPARQL query is generated based on the ontology triples. This is done by the QueryGenerator module, that combines constant values with the generated ontology triples in order to create the correct SPARQL query. These queries take the form shown in Figure 5. The starting variables (for the generated pattern graph) are used in the SELECT clause of the SPARQL query.

**Processing Query** After the query is generated, it needs to be processed in order to retrieve any results. We use ARQ to process the queries and retrieve the results from the ontology. ARQ is a query processor for SPARQL that was developed to work with ontologies that are already loaded in Jena. The query processor returns a result set from the ontology, which our engine alters so that it represents a list of news items or a direct answer from the knowledge base.

The results of the processed query are displayed as a list of news items, where each news item is shown together with the concept that triggered the inclusion of that particular item in the results set. At the top of the screen, the user can again exclude or include concepts in order to restrict the results set. Figure 6 illustrates this step.

```
   input  : A parse tree parseTree which was returned by the Stanford Parser
   output: A list ontologyTriples which is filled with the ontology triples
 1 currentTerm ← "news items";
 2 keywords ← "mention, concern, related to";
 3 if parseTree.contains (currentTerm) then
 4     currentTerm.update ();
 5     ontologyTriples.add (rdfTypeTriple(?news, News);
 6     if hasLink (currentTerm, keywords) then
 7         currentTerm ← keywords;
 8         ontologyTriples.add (relationTriple(?news, hermes:relation, ?relation));
 9         ontologyTriples.add (rdfTypeTriple(?relation, Relation));
10         if hasLink (currentTerm, ontologyConcept) then
11             currentTerm ← ontologyConcept;
12             ontologyTriples.add (relationTriple(?relation, hermes:relatedTo,
               ?concept));
13             ontologyTriples.add (rdfTypeTriple(?concept,
               getOntologyClass(currentTerm)));
14             previousTerm ← currentTerm;
15             while more linked concepts exist do
16                 if hasLink (currentTerm, ontologyConcept) then
17                     currentTerm ← ontologyConcept;
18                     ontologyTriples.add (relationTriple(?concept,
                       getRelation(previousTerm, currentTerm),
                       ?anotherConcept));
19                     ontologyTriples.add (rdfTypeTriple(?concept,
                       getOntologyClass(currentTerm)));
20                 end
21             end
22         end
23     end
24 else
25     if parseTree.contains (ontologyConcept) then
26         currentTerm ← ontologyConcept;
27         ontologyTriples.add (rdfTypeTriple(?concept,
           getOntologyClass(currentTerm)));
28         previousTerm ← currentTerm;
29         while more linked concepts exist do
30             if hasLink (currentTerm, ontologyConcept) then
31                 currentTerm ← ontologyConcept;
32                 ontologyTriples.add (relationTriple(?concept,
                   getRelation(previousTerm, currentTerm), ?anotherConcept));
33                 ontologyTriples.add (rdfTypeTriple(?concept,
                   getOntologyClass(currentTerm)));
34             end
35         end
36     end
37 end
38 return ontologyTriples;
```

**Fig. 3.** Developed heuristics for generating the ontology triples from the parse tree

```
(?company rdf:type hermes:Company)
(?company hermes:hasCEO ?ceo)
(?ceo rdf:type hermes:Business_leaders)
(?ceo hermes:name "Steve Ballmer")
```

**Fig. 4:** "Which company has Steve Ballmer for CEO?": ontology triples

```
PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
PREFIX news: <http://www.jborsje.nl/hermes/news.owl#>
PREFIX hermes: <http://www.jborsje.nl/hermes/news.owl#>
SELECT DISTINCT ?company
WHERE {
  ?company rdf:type hermes:Company .
  ?company hermes:hasCEO ?ceo .
  ?ceo rdf:type hermes:Business_leaders .
  ?ceo hermes:name "Steve Ballmer" .
}
```

**Fig. 5:** "Which company has Steve Ballmer for CEO?": SPARQL query

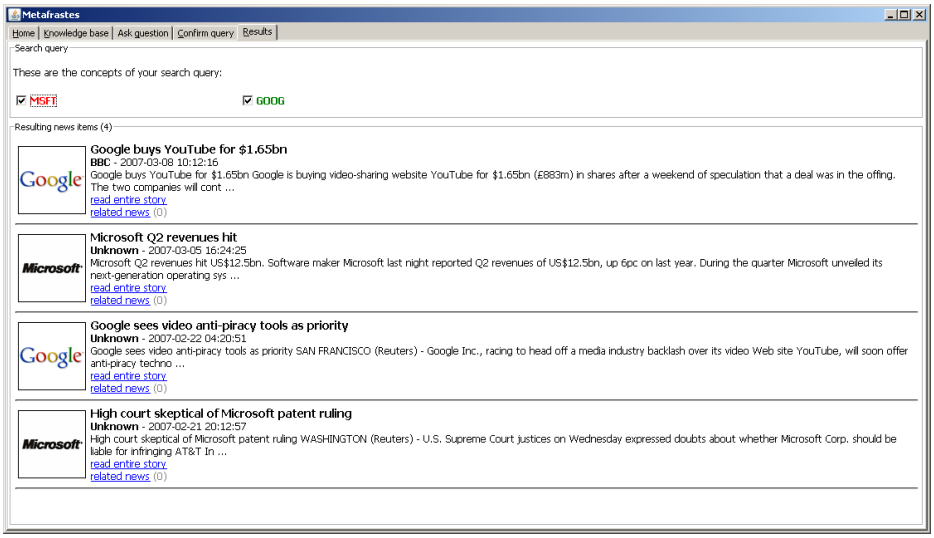


Fig. 6: Displaying the results in METAFRASTES

## 4 Evaluation

Evaluating the results that are produced by METAFRASTES is equivalent to evaluating the generated SPARQL queries, as these queries ultimately provide the results. Thus, the quality of the results depends on the quality of the generated SPARQL query. The major part of the generated SPARQL query consists of the

ontology triples that were generated earlier by the triple generator. Thus, the quality of the generated SPARQL query depends on the quality of the ontology triples that are produced by the triple generator. Consequently, we evaluate the performance of METAFRASTES by evaluating the ontology triples that are produced in order to come to a conclusion on the quality of the results.

The evaluation of the generated SPARQL queries, as well as the quality of the generated results in terms of soundness and completeness, is done by the authors. It should be noted that this evaluation is objective, due to the crisp nature of the responses to the posed queries. At this stage we did not perform any evaluation of the system based on other criteria such as ease of use of the system/user experience.

### 4.1 Evaluating Example Questions

To get to the evaluation results as described in this section, we did one run in which we asked 9 example questions (assuming that the ontology structure is knows). The resulting ontology triples were examined for correctness and, based on this, it was decided whether the NLP system produced an accurate translation of the question. We consider a query to be correctly translated when the results of the SPARQL query are sound and complete with respect to the original natural language query (completeness is considered in relation to the information stores in the ontology). The example questions are reproduced in Table 2, along with their translation scores.

**Table 2.** Selected example user questions and their translation scores

| **Question** | **Translation** |
|---|---|
| Which news items mention Google? | Correct. |
| Which companies are competitors of Google? | Correct. |
| In which news items does at least one of Google's competitors appear? | Incorrect. |
| Who is the CEO of Apple? | Correct. |
| Which news items mention the CEO of Apple? | Correct. |
| Which company produces iPhones? | Correct. |
| Which news items mention the company that produces iPhones? | Correct. |
| What is the name of the CEO of the company that produces iPhones? | Correct. |
| Which company has Steve Ballmer for CEO? | Correct. |

We can see from this table that 8 out of 9 questions were translated correctly. We can conclude from this that our NLP engine performs quite well on the set of example questions.

### 4.2 Evaluating Custom Questions

When evaluating the system based on a set of custom questions, we applied the same procedure as the one we described in the previous section. Table 3 shows a

list of 9 custom questions. They are questions that we came up with, regardless of the structure of the ontology. Moreover, we tried to act a bit like 'layman users' ourselves, resulting in questions that are not always formulated in a clear way.

**Table 3.** Some custom user questions and their translation scores

| # | **Question** | **Translation** |
|---|---|---|
| C1 | When was YouTube bought by Google? | Incorrect. |
| C2 | Which companies have recently experienced a CEO switch? | Incorrect. |
| C3 | Give me the names of all software companies. | Incorrect. |
| C4 | Is Google situated in the Netherlands? | Incorrect. |
| C5 | Does Larry Page work at Microsoft? | Incorrect. |
| C6 | Which CEO works for Adobe? | Correct. |
| C7 | iPhones are great. Where can I get one? | Incorrect. |
| C8 | Who is the CEO of Motion in Research? | Correct.[1] |
| C9 | What does Apple do? | Incorrect. |

Table 3 shows that our NLP engine performs poorly when it comes to questions which are not as structured as our example questions. In some cases the incorrect translation followed from the fact that the question was formulated in a vague manner(such as question C7 and C9). Some questions were properly formulated, but just too complex for the NLP engine to translate properly.

## 5 Conclusions & Future Work

The work we present here enables us to draw some conclusions on ontology-based information querying using NLP. We conclude that as the use of Semantic Web technologies are becoming more popular, they are also used more frequently in various NLP systems. Our work also shows that these technologies can greatly support NLP through their data representation capabilities. Moreover, our work shows that current technologies provide a good foundation for new systems that are able to bridge the gap between expert- and layman users.

As future work we will focus on improving the quality of the NLP process. Also, we will focus on enabling users of the system to use custom ontologies rather than the domain ontology provided by the system. In the evaluation we would like to consider more complex queries, such as, for example, queries containing constraints on the time frame of news items.

[1] The 'Correct' score was assigned because no answer was given; the answer was omitted because the system recognised the use of concepts that were not present in the knowledge base and as a result threw a proper ConceptNotFoundException.

## References


1. ARQ, a SPARQL Processor for Jena. http://jena.sourceforge.net/ARQ/.
2. Hermes – Semantic News Portal. http://hermesportal.sourceforge.net/.
3. R. Albert, H. Jeong, and A.-L. Barabasi. The Diameter of the World Wide Web. *Nature*, 401:130–131, 1999.
4. I. Androutsopoulos, G. Ritchie, and P. Thanisch. Masque/SQL – An Efficient and Portable Natural Language Query Interface for Relational Databases. In *Proceedings of the 6th International Conference on Industrial & Engineering Applications of Artificial Intelligence and Expert Systems*, pages 27–30. Gordon & Breach Science Publishers, 1993.
5. T. Berners-Lee, J. Hendler, and O. Lassila. The Semantic Web. *Scientific American*, 284(5):34–43, 2001.
6. D. Brickley and R. Guha. RDF Vocabulary Description Language 1.0: RDF Schema. *W3C Recommendation 10 February 2004*, February 2004. http://www.w3.org/TR/rdf-schema/.
7. N. Guarino and C. Welty. Evaluating ontological decisions with ontoclean. *Communications of the ACM*, 45(2):61–65, 2002.
8. S. Harris and A. Seaborne. Sparql 1.1 query language. *W3C Working Draft*, 14, 2010.
9. A. Java, T. Finin, and S. Nirenburg. SemNews: A Semantic News Framework. In *Proceedings of the 21st National Conference on Artificial Intelligence (AAAI-06)*. American Association of Artificial Intelligence, February 2006.
10. A. Java, T. Finin, and S. Nirenburg. Text Understanding Agents and the Semantic Web. In *Proceedings of the 39th Hawaii International Conference on System Sciences (HICSS 2006)*. IEEE Computer Society, January 2006.
11. S. J. Kaplan. Designing a Portable Natural Language Database Query System. *ACM Transactions on Database Systems*, 9(1):1–19, 1984.
12. D. Klein and C. D. Manning. Accurate Unlexicalized Parsing. In *Proceedings of the 41st Meeting of the Association for Computational Linguistics*, pages 423–430, 2003.
13. G. Klyne and J. J. Caroll. Resource description framework (RDF): Concepts and abstract syntax. *W3C Recommendation 10 February 2004*, February 2004. http://www.w3.org/TR/rdf-cocnepts/.
14. V. Lopez, M. Pasin, and E. Motta. AquaLog: An Ontology-Portable Question Answering System for the Semantic Web. *European Semantic Web Conference (ESWC 2005)*, pages 135–166, 2005.
15. B. McBride. Jena: A Semantic Web Toolkit. *IEEE Internet Computing*, 6(6):55–59, 2002.
16. D. McGuinness and F. van Hamelen. OWL Web Ontology Language. *W3C Recommendation 10 February 2004*, February 2004. http://www.w3.org/TR/owl-features/.
17. V. Milea, F. Frasincar, and U. Kaymak. tOWL: A temporal web ontology language. *Systems, Man, and Cybernetics, Part B: Cybernetics, IEEE Transactions on*, 42(1):268–281, 2012.
18. V. Milea, F. Frasincar, U. Kaymak, and G. Houben. Temporal optimisations and temporal cardinality in the tOWL language. *International Journal of Web Engineering and Technology*, 7(1):45–64, 2012.
19. E. Prud'hommeaux and A. Seaborne. SPARQL Query Language for RDF. *W3C Recommendation January 2008*, January 2008. http://www.w3.org/TR/rdf-sparql-query/.