Pertanika J. Sci. & Technol. 21 (1): 193 - 204 (2013)

PERTANIKA JOURNALS

SCIENCE & TECHNOLOGY

Journal homepage: http://www.pertanika.upm.edu.my/

A Negation Query Engine for Complex Query Transformations

Rizwan Iqbal* and Masrah Azrifah Azmi Murad

Faculty of Computer Science and Information Technology, Universiti Putra Malaysia, Serdang, Selangor, Malaysia

ABSTRACT

Natural language interfaces to ontologies allow users to query the system using natural language queries. These systems take natural language query as input and transform it to formal query language equivalent to retrieve the desired information from ontologies. The existing natural language interfaces to ontologies offer support for handling negation queries; however, they offer limited support for dealing with them. This paper proposes a negation query handling engine which can handle relatively complex natural language queries than the existing systems. The proposed engine effectively understands the intent of the user query on the basis of a sophisticated algorithm, which is governed by a set of techniques and transformation rules. The proposed engine was evaluated using the Mooney data set and AquaLog dataset, and it manifested encouraging results.

Keywords: Natural language interfaces, ontology, semantic web, negation queries, search engines

INTRODUCTION

Many natural language interfaces have been developed to date. The concept of natural language interfaces is not new. Natural language interfaces were initially used for databases which allowed users to submit their queries in natural language instead of writing in SQL query format. Since the emergence of ontologies in the field of computer science, researchers have started developing interfaces for them. Natural language interfaces facilitate users to express their information needs in natural language that they are familiar with and can consequently populate knowledge bases.

Depending upon the ability to process natural language input, natural language interface can be classified into two categories, namely full natural language interface and restricted natural language interface. Full natural language interfaces provide the ease of inputting a natural language query without any

Article history:
Received: 31 March 2012
Accepted: 31 August 2012

E-mail addresses:
mail.rizwaniqbal@yahoo.com (Rizwan Iqbal),
masrah@fsktm.upm.edu.my (Masrah Azrifah Azmi Murad)
*Corresponding Author

restriction of vocabulary, whereas restricted natural language interfaces contradict the former and restrict users to input a restricted vocabulary.

Natural language interfaces work by converting natural language into formal semantic query (Tablan *et al*., 2008a). Different interfaces rely on distinct techniques and algorithms to translate natural language query into formal semantic query. Some natural language interfaces that support full natural language support are Semsearch (Lei *et al*., 2006), NLP-Reduce (Kaufmann *et al*., 2007), FREya (Damljanovic *et al*., 2010), QuestIO (Tablan *et al*., 2008b), Spark (Zhou *et al*., 2007), Q2Semantic (Wang *et al*., 2008) and NLION (Ramachandran & Krishnamurthi, 2009). Natural language interface that comes under the category of restricted natural language interfaces include Orakel (Cimiano *et al*., 2008), and AquaLog (Vanessa *et al*., 2007).

Natural language queries with negation are a very usual and expected input from the user. In this study, Mooney dataset (MooneyData, 1994), which is widely applied for the evaluation of natural language interfaces and systems (Wang *et al*., 2007; Damljanovic *et al*., 2010; Tablan *et al*., 2008b; Kaufmann *et al.*, 2006; Iqbal *et al.,* 2012) was used. This dataset has a number of negation queries in it. This paper proposes a negation query handling engine, which effectively caters negation natural language queries. The proposed negation query handling engine supports full natural language rather than restricted language. The proposed engine effectively understands the intent of the user's negation query on the basis of a sophisticated algorithm, which is governed by a set of techniques and transformation rules. The rest of this paper is organized as follows. Work related to the current study is discussed in the next section. Then, the motivation for the proposed negation query handling engine is discussed. Later sections cover the design, technical details and evaluation of the proposed engine. Finally, the paper is concluded by the conclusion and future work section.

RELATED WORK

All developed natural language interfaces differ from each other in one way or another. Some natural language interfaces focus on giving users the freedom of entering natural language query using free vocabulary (Lei *et al*., 2006), while others allow users to enter natural language query using a restricted vocabulary (Cimiano *et al*., 2008; Vanessa *et al*., 2007). Other than natural language interfaces, there are also interfaces that allow the user to search knowledge bases by inputting formal language queries. Such search engines are of two types; first is the form-based search engine, which provides web forms as a means of specifying queries (Rocha *et al.,* 2004), and second is the RDF-based querying search engine which supports RDF-based querying languages at the front end (Rocha *et al*., 2004).

SHOE (Heflin & Hendler, 2000) is an interface which comes in the category of form-based semantic search engine. This system is based on a semantic mark-up language. SHOE (Heflin & Hendler, 2000) allows users to define and associate vocabularies which can be understandable by the machines. The system uses knowledge annotator to add up SHOE annotations to the documents. Naive users often feel uncomfortable using SHOE as the web forms require familiarity with the knowledge bases being searched. On the other hand, adding annotations with SHOE is more time consuming as compared to standard XML.

Corese (Corby *et al*., 2004) search engine is from the category of RDF-based querying language fronted search engines. This search engine is dedicated to RDF metadata. The Corese engine internally works on the concept of conceptual graphs (CG). The query and the ontology schema are translated from RDF to conceptual graphs (CG) in order to perform matching operations. The limitation with such search systems is that they require the users to be familiar with both the knowledge base and the querying language used.

The kind of interfaces that take a formal language query for input remains comparatively less preferred by users than the natural language query interfaces. The reason is the fact that users require training of the system as well as reasonable knowledge about the knowledge bases being searched. When talking about the systems that support natural language, SemSearch (Lei *et al*., 2006) is an interface that was designed to take input in natural language from the users. The system has idealized Google for the style of its interface. In particular, SemSearch (Lei *et al*., 2006) uses three heuristic operators to support its search. The use of these heuristic operators helps the system to have a clue of what information exactly the user wants from the knowledge bases. SemSearch relies on simple string matching algorithms rather than using complex techniques like WordNet (Fellbaum, 1998) in order to reduce the response time of the system. The use of the simple string matching algorithms reduces the response time but at the same time, it can be a reason for losing some good matches in certain situations.

AquaLog (Vanessa *et al*., 2007) is a natural language based question-answering system. It takes input in the natural language query and answers on the basis of ontology loaded in the system. AquaLog has a learning mechanism in which it helps to improve the performance over time. Relation Similarity Service (RSS) is a component of AquaLog and it is considered as the backbone of the system. In case of any ambiguity between multiple terms, the RSS module directly interacts with the user to disambiguate between the terms of natural language and the concepts of the knowledge bases. The limitation with AquaLog is that it can at the most translate the query to two triples.

QuestIO (Tablan *et al*., 2008b) is another natural language interface that was designed to take natural language input from the user. It was designed to cater language ambiguities, handle incomplete and grammatically incorrect queries. QuestIO focuses to be an open domain system that does not require any customization by the users, as well as any training to use it. This system uses light weight linguistic processing that allows the user's text to be fully analyzed in the query processing part for the identification of ontology concepts and property names. QuestIO works by finding implicit relations which are not clearly stated in the user query. The limitation is that it only works with directly explored relations (Tablan *et al*., 2008b).

The creators of QuestIO (Tablan *et al*., 2008b) later developed another natural language interface named FREya (Damljanovic *et al*., 2010). The focus of creating FREya was to further reduce customization efforts and to introduce clarification dialogues mechanism to avoid empty results. The clarification dialogue mechanism in FREya helps users to get answers in case the system is not able to find any answer (Damljanovic *et al*., 2010). If the system does not come up with an answer automatically, it will interact with the end users to get a clue for the right answer. The user's selections are saved over time and the system learned to place correct suggestions on top of any similar query next time on the basis of the saved user selections. FREya reported satisfactory results for the learning mechanism in it but its correctness without clarification dialogues was considerably low as compared to other similar systems like PANTO (Wang *et al*., 2007) for the same data. Damljanovic *et al*. (2010) also reported that FREya was not able to answer some questions correctly while evaluating. The questions that were answered incorrectly included those with negation. PANTO (Wang *et al*., 2007) is another natural language interface that is portable and does not make any assumption about any specific knowledge domain. It functions in a way that it picks the words from the natural language query and map them to entities (concepts, instances, relations) in the ontology.

The architecture of PANTO (Wang *et al*., 2007) relies on the existing tools like WordNet (Fellbaum, 1998) and different string metric algorithm. In PANTO, the Lexicon Builder automatically extracts ontological resources (Classes, Object and Datatype properties, Literals, Instances) from the ontology and constructs the Lexicon. The translator is the core processing engine of PANTO. It receives the processed natural language query from the parser as input and then performs operations to map the natural language entities to the ontological entities.

PANTO uses an off the shelf parser which relies on limited NLP techniques, and this restricts the scope of queries which can be handled by the system (Wang *et al*., 2007). Wang *et al.* (2007) also discussed the limitation of PANTO in relation to the weakness in supporting complex user interactions. The current version of PANTO deals with superlative, comparative, conjunction and negation kind of queries. Nonetheless, PANTO has not discussed how effectively it can deal with the queries. PANTO discusses that it can handle negation queries including "not" and "no". However, Wang *et al.* (2007) have not discussed in detail how effectively PANTO can deal with negation queries or what the precision of the system is in catering particularly with negation queries. From the literature, it is found that all systems have discussed about their supports for catering negation queries. Wang *et al.* (2007) mentioned that it could support negation queries but did not give any detail that to what extent and precision it could cater them. Damljanovic *et al.* (2010) discussed about catering negation cases, and explicitly mentioned that their system had failed to answer some questions correctly and amongst them were questions with negation.

MOTIVATION FOR NEGATION QUERY HANDLING ENGINE

After studying the trend in some natural language interfaces over the years, it is found that the state-of-art interfaces are particularly focusing on some specific features. Some of these features include portability (Wang *et al*., 2007), users' interaction (Damljanovic *et al*., 2010), automated learning (Vanessa *et al*., 2007; Damljanovic *et al*., 2010), lesser customization (Lei *et al*., 2006), detecting and resolving the ambiguity in natural language (Lei *et al*., 2006) and precision in understanding the complexity of the natural language query (Wang *et al*., 2007; Tablan *et al*., 2008a). For a natural language interface to be effective for the user, it is very critical for it to understand the complexity of the natural language query inputted by the user. Over the years, researchers have improved from simple string matching algorithms to advance natural language processing engines which are designed to understand the complexity of natural language.

It was found that natural language interfaces have the capability to deal with different types of queries such as instance superlative, comparative, conjunction and negation. Systems like

PANTO (Wang *et al*., 2007) and FREya (Damljanovic *et al*., 2010) have discussed negation queries. In particular, PANTO has given no details about the extent, details and precision for which it can handle the negation queries. FREya only discussed that the system was not able to answer some questions correctly and amongst them were questions with negation. Natural language queries with negation are very usual and expected input from the user. A review of relevant literature has shown that the Mooney dataset (MooneyData, 1994) and AquaLog dataset (AquaLogData, 2007) are widely used for the evaluation of natural language interfaces (Wang *et al*., 2007; Damljanovic *et al*., 2010; Tablan *et al*., 2008a; Kaufmann *et al*., 2007). These datasets have negation queries in them.

A natural language interface with the capability to handle negation queries must be able to correctly interpret the intent of the user's query to its equivalent formal query language. In order to correctly interpret the intent of the user's query, the natural language interface cannot just rely on simple keyword detection and simple string matching algorithms. An effective negation handling interface must have a sophisticated algorithm that is governed by a set of rules. These rules should give a deeper insight into the natural language interface about the negation query under consideration. An in-depth understanding of the negation query will facilitate the interface to make appropriate transformations of natural language query to its equivalent formal query language, keeping intact the intent of the user.

The absence of such an algorithm focusing to handle negation queries has become a motivation for this research. This research came up with a negation query handling engine which incorporates an algorithm that has been particularly designed to cater negation queries in an effective manner. The next section discusses the design of negation query handling engine in detail.

THE DESIGN OF NEGATION QUERY HANDLING ENGINE

The negation query handling engine was designed to cater effective machine level transformation for the natural language query entered by the user. The engine was designed to perform some processes in a sequential manner. There is also a set of natural language query transformation rules which are implemented according to the structure of the natural language query entered by the user. Fig.1 shows how the negation query handling engine works.

The processes performed by the negation query handling engine

The negation query handling engine performs three processes on the natural language query entered by the user. The first process identifies the negation keywords in the natural language query. The second process is responsible to identify the coordinating conjunctions in the natural language query structure. The final process is responsible to make sure that the natural language keywords are correctly mapped to the corresponding ontological resources (Classes, Object and Datatype properties, Literals, Instances) and the machine level transformations are correctly carried out.

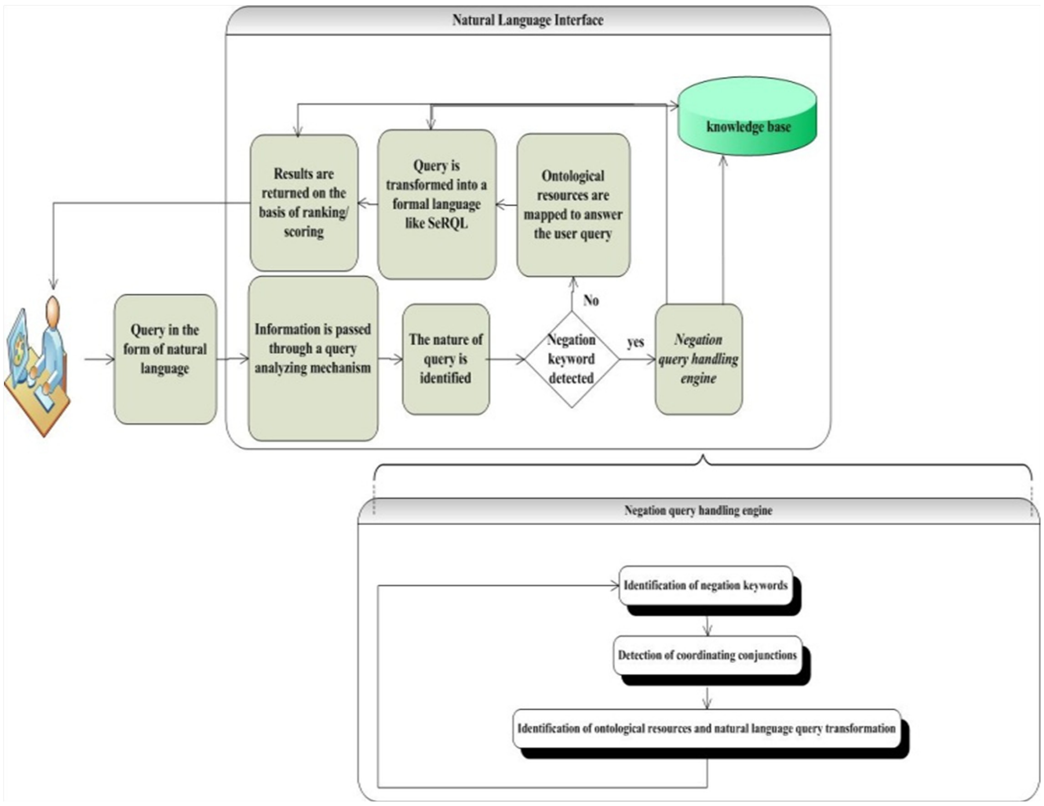
Natural Language Interface
knowledge base
Results are returned on the basis of ranking/ scoring
Query is transformed into a formal language like SeRQL
Ontological resources are mapped to answer the user query
No
Query in the form of natural language
Information is passed through a query analyzing mechanism
The nature of query is identified
Negation keyword detected
yes
Negation query handling engine
Negation query handling engine
Identification of negation keywords
Detection of coordinating conjunctions
Identification of ontological resources and natural language query transformation

Fig.1: Negation query handling engine

Identification of negation keywords

This process deals with the identification of negation keywords. The engine intelligently identifies keywords with a broad coverage in the aspect of negation. Some previous systems like PANTO (Wang *et al*., 2007) suffer limited support for negation queries. In particular, PANTO handles those queries including "not" and "no". This engine was designed to handle several kinds of queries including those with does not/ do not/ don't/ excluding/ except/ leaving/ none/ and other than. The proposed engine also has provision to handle affirmative-negative and affirmative-negative-pseudorel type of queries, whereby such queries can be found in the AquaLog data set (AquaLogData, 2007). In addition, the proposed engine does not only identify negation keywords but also makes appropriate transformations from the natural language to formal query language (like SPARQL) keeping in integrity the negation sense of the user query.

Detection of coordinating conjunctions

After the occurrence of negation keywords, the engine looks for coordinating conjunctions in the natural language query. Coordinating conjunctions include for, and, nor, but, or, yet, so. If a coordinating conjunction is detected, only the keywords before it will be considered for query formation. The occurrence of a coordinating conjunction depicts that the query has more than one condition in it. For example, if the Mooney dataset (MooneyData, 1994) which is used as a test data for many natural language interfaces to ontologies is considered, it is found that there are many queries in it which have more than one condition in them. The existence of more than one condition in the user's query makes query transformation a complex task especially when handling negation queries.

Identification of ontological resources and natural language query transformation

This process deals with the identification of ontological resources. After the identification of ontological resources (Classes, Object and Datatype properties, Literals, Instances), the engine performs the transformation of natural language query to its formal query language equivalent. The engine ensures that the transformation is in alignment with the viewpoint of the user. The natural language query transformation rules were designed on the basis of some rules. These rules have been found to be satisfying the negation queries within the Mooney dataset (MooneyData, 1994) and AquaLog dataset (AquaLogData, 2007).

The natural language query transformation rules

The negation query handling engine transforms natural language query to formal query language on the basis of some rules. These rules are applied to the natural language query, depending on the detection of certain scenarios. These rules give an insight into the engine about the intent of the user query and actions which should be applied on the user's query for transforming it into the formal query language equivalent. These rules were tested on the Mooney dataset and AquaLog dataset, and encouraging results were seen while performing the query transformation. Table 1 shows some query transformation rules.

Step-by-step processing of the natural language query

The negation query handling engine performs step-by-step operation on the natural language query inputted by the user. The operations on the natural language query can handle a negation query with two possible cases. The first case deals with explicit negation words like not/ do not/ don't/ excluding/ except/ leaving/ none. The second case deals with affirmative-negative and affirmative-negative-pseudorel type of queries. Below is the step-by-step processing for the first case.

1. A negation keyword is detected in the user's query.
2. After detecting the negation keyword, the engine looks for a coordinating conjunction.
3. If a coordinating conjunction is found, only the keywords before it are considered for query transformation; otherwise, all the keywords are considered for the query transformation.
4. The engine tries to map the literals and instances from the knowledge base to the keywords in the user's query.
5. If the engine fails to find an appropriate match for a literal or an instance, it is then expected that a data or object be detected.

TABLE 1: Query transformation rules

Rules	Actions
Coordination conjunction detected.	The query transformation is not based on a single step. The keywords before the coordinating conjunction are considered for the first step of the natural query transformation. The remaining part of the natural query is treated as the second part of a query transformation. After the completion of the transformation for the first part of the query, the remaining part of the query will be treated as a different part and all the transformation processes will be performed from the beginning.
Coordination conjunction is not detected.	The query transformation is a single step. All the keywords in the natural language query are considered for the natural language query transformation.
Literal or instance is not detected.	The natural language query refers to an object or data property. Matching algorithms are run to find the appropriate matches for the object or data property from a list of ontological resources.
Literal or instance is detected.	The natural language query has detected a literal or an instance. The next step is to find the associated data or object property with the detected literal or instance from a list of ontological resources.
Is/ Does/ Has is detected at the beginning of the Query.	This is the condition that deals with affirmative-negative and affirmative-negative-pseudorel type of queries. The next step is to detect an instance in between Is/ Does/ Has and "?" in the query. If an instance is detected, all the keywords (excluding Is/ Has/ Does) and the instance will then be matched to find the best match for object property associated with the instance.

6. Transformations are based on some predefined rules, depending whether a literal/ instance or a data/object property is detected.
7. If there is a coordinating conjunction detected at step 3, all the steps will be repeated for the remaining part of the query.

The step-by-step processing for the second case is as below.

1. Is/ Does/ Has is detected at the beginning of the query which ends with a "?".
2. The next step is to look for an instance after the words Is/ Does/ Has.
3. If an instance is found, the next step is to find an object property.
4. Matching is performed by the algorithm, which excludes Is/ Does/ Has, and the instance is to find the best match for the object property associated with the instance.
5. Transformations are performed based on some predefined rules and templates.

EVALUATION OF NEGATION QUERY HANDLING ENGINE

The designed negation query handling engine is evaluated using the negation queries in the Mooney dataset (MooneyData, 1994) and Aqualog dataset (AquaLogData, 2007). The Mooney data set included a total of 88 negation queries. There are 74 negation queries in the job Mooney dataset and 14 negation queries in the geography Mooney dataset. There are a total 24 negation queries in the Aqualog dataset. The Aqualog dataset has divided these negation queries into three different categories. The category names are affirmative-negative, affirmative-negative-pseudorel and negation. The designed algorithm of the negation query handling engine correctly transformed 72.7% of the negation queries in the Mooney dataset and 41.6% of the negation queries in the Aqualog data set to their formal query language equivalent. Table 2 and Fig.2 show the evaluation of the proposed algorithm.

The designed algorithm of the negation query handling engine manifested encouraging results for the negation queries in the Mooney dataset and AquaLog dataset. All the negation queries in the Mooney dataset and AquaLog data set were parsed through the algorithm of the proposed negation query handling engine. Below are some examples of how every query with negation is individually parsed and the proposed algorithm is evaluated.

Sample Query 1 (Job Mooney data set): **"Are there ada jobs outside austin?"**

Refer to Fig.3 for details of the sequence of processing natural language in the proposed negation query handling engine.

Step 1 : Negation keyword detected: **outside.**

Step 2 : Coordinating conjunction not detected. According to the transformation rules, all the words in the natural language query will be considered for formal language query transformation (SPARQL).

Step 3 : Instances are detected: **"ada"** and **"austin".** According to the query transformation rules (Table 2) if a literal or an instance is detected, the next step is to find the associated data or object property with the detected literal or an instance from the list of ontological resources.

Step 4 : The negation query handling engine will look for the object properties associated with the instances. The engine will find the associated object properties associated with

TABLE 2: Results obtained for query transformation

Dataset	Total negation queries	No of queries correctly transformed to formal query language	% of queries correctly transformed to formal query language
Mooney	88	64	72.7%
Aqualog	24	10	41.6%

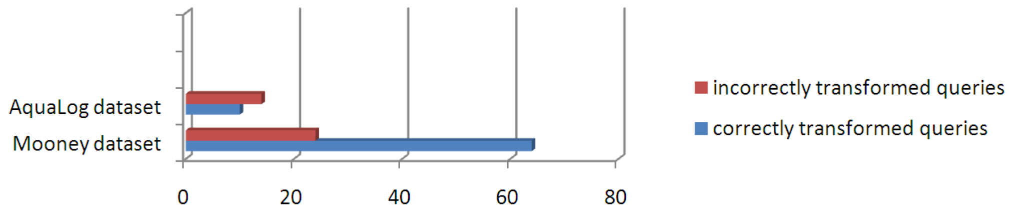

Fig.2: Query transformation results

“ada” and **“austin”**. As for **“ada”**, the system will locate **“useslanguage”** and for **“austin”**, the system will recognize **“isinCity”**.

Step 5: The selected object properties are set into pre-defined template in SPARQL in a specified format. The basic format is as below. The selected object property is set in the following sequence of triple.

?subject name of selected property?Var

FILTER(?Var!=(name of instance or literal))

In the case of the exampled query, the exact query transformation in SPARQL will be in the following format. The negation query handling engine will intelligently identify the intent of the user in the negation sense and will place a “!” in the filter part of the SPARQL query. In the case of the example query, the “?City!=p1:austin” means to exclude all those jobs which are not in “Austin” city.

SELECT ?subject ?Lang ?City

WHERE{?subject a p1:ITJob.

?subject p1:usesLanguage ?Lang.

?subject p1:isInCity ?City.}

FILTER(?Lang=p1:ada && ?City!=p1:austin).

CONCLUSION AND RECOMMENDATIONS FOR FUTURE WORK

This paper has proposed a negation query handling engine that was specifically designed for handling negation queries. The designed negation query handling engine was evaluated using the Mooney dataset (MooneyData, 1994) and AquaLog dataset (AquaLogData, 2007). It was found that the proposed negation query handling engine correctly transformed 72.7% of Mooney dataset and 41.6% of AquaLog dataset negation queries to their formal query language equivalent. The proposed engine demonstrated encouraging results. Hence, it is a step forward towards an effective handling of complex natural language query transformations.

It is an intention for the future to further improve the algorithm so as to bring more correctness in the negation query transformations. An evaluation of the proposed engine on other datasets may also lead to new insights, which can be incorporated in the existing algorithm of the negation query handling engine to make it more effective and robust in handling negation queries.

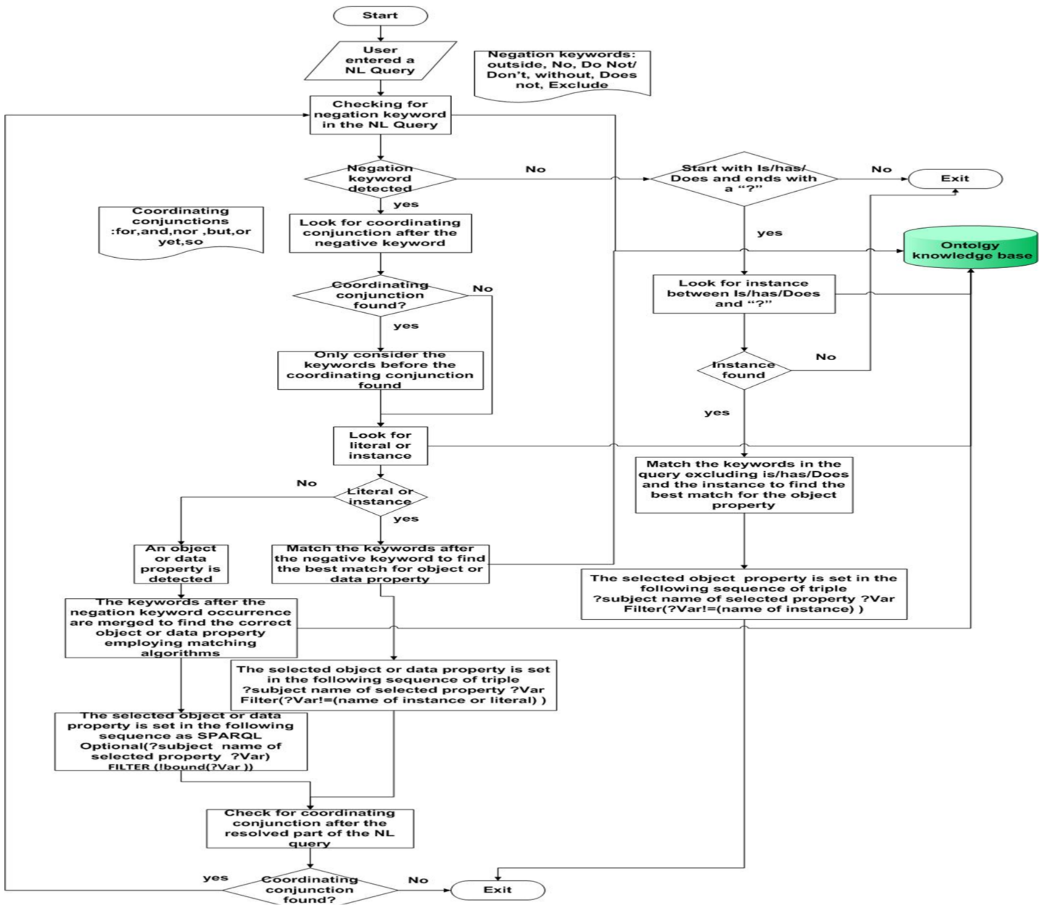
Start
User entered a NL Query
Negation keywords: outside, No, Do Not/ Don't, without, Does not, Exclude
Checking for negation keyword in the NL Query
Negation keyword detected
No
Start with Is/has/ Does and ends with a "?"
No
Exit
yes
Coordinating conjunctions :for,and,nor ,but,or yet,so
Look for coordinating conjunction after the negative keyword
yes
Ontolgy knowledge base
Coordinating conjunction found?
No
Look for instance between Is/has/Does and "?"
yes
Only consider the keywords before the coordinating conjunction found
Instance found
No
yes
Look for literal or instance
Match the keywords in the query excluding is/has/Does and the instance to find the best match for the object property
No
Literal or instance
yes
An object or data property is detected
Match the keywords after the negative keyword to find the best match for object or data property
The selected object property is set in the following sequence of triple ?subject name of selected property ?Var Filter(?Var!=(name of instance))
The keywords after the negation keyword occurrence are merged to find the correct object or data property employing matching algorithms
The selected object or data property is set in the following sequence of triple ?subject name of selected property ?Var Filter(?Var!=(name of instance or literal))
The selected object or data property is set in the following sequence as SPARQL Optional(?subject name of selected property ?Var) FILTER (!bound(?Var))
Check for coordinating conjunction after the resolved part of the NL query
yes
Coordinating conjunction found?
No
Exit

Fig.3: The step-by-step processing of the natural language query

REFERENCES

AquaLogData. (2007). Retrieved from http://kmi.open.ac.uk/technologies/aqualog/examples.html

Cimiano, P., Haase, P., Heizmann, J., Mantel, M., & Studer, R. (2008). Towards portable natural language interfaces to knowledge bases - The case of the ORAKEL system. *Data and Knowledge Engineering*, *65*, 325-354.

Corby, O., Dieng-Kuntz, R., & Faron-Zucker, C. (2004). Querying the Semantic Web with the corese search engine*. 16th European conference on Artificial Intelligence* (ECAI-2004), 705-709.

Cohen, W. W., Ravikumar, P., & Fienberg, S. E. (2003). A Comparison of String Distance Metrics for Name-Matching Tasks. *Workshop on information integration on the Web* (IIWeb-03), 73-78.

Damljanovic, D., Agatonovic,M., & Cunningham, H. (2010). Natural Language interface to ontologies: Combining syntactic analysis and ontology-based lookup through the user interaction. *European Semantic Web Conference* (ESWC 2010), 106-120.

Fellbaum, C. (1998). WORDNET: An Electronic Lexical Database. *MIT Press*, 1998.

Heflin, J., & Hendler, J. (2000). Searching the web with SHOE. *Workshop on AI for Web Search* (AAAI-2000).

Iqbal, R., Murad, M. A. A., Selamat, M. H., & Azman, A. (2012). Negation query handling engine for natural language interfaces to ontologies. *International Conference on Information Retrieval and Knowledge Management (CAMP'12)*, 249-253.

Kaufmann, E., Bernstein, A., & Fischer, L. (2007). Nlp-Reduce: A "naïve" but domain independent natural language interface for querying ontologies. *European Semantic Web Conference* (ESWC 2007).

Kaufmann, E., Bernstein, A., & Zumstein, R. (2006). Querix: A Natural Language Interface to Query Ontologies Based on Clarification Dialogs. *International Semantic Web Conference* (ISWC 2006), Springer, 980-981.

Lei,Y., Uren, V., & Motta, E. (2006). Semsearch: a search engine for the semantic web. *Managing Knowledge in a World of Networks*, *4248*, 238-245.

Motro, A. (1986). Constructing queries from tokens. *SIGMOD international conference on Management of data* (SIGMOD '86), 120-131.

MooneyData. (1994). Retrieved from http://www.cs.utexas.edu/users/ml/nldata.html.

Rocha, C., Schwabe, D., & de Aragao, M. (2004). A Hybrid Approach for Searching in the Semantic Web. *13th International conference on World Wide Web* (WWW'04), 374-383.

Ramachandran, V. A., & Krishnamurthi, I. (2009). NLION: Natural Language interface for querying ontologies. *2nd Bangalore Annual Compute Conference* (COMPUTE' 09), 1-4.

Tablan,V., Damljanovic, D., & Bontcheva, K. (2008a). A natural language query interfaceto structured information. *5th European semantic web conference on The semantic web: research and applications* (ESWC'08), Springer Berlin/Heidelberg, LNCS, 5021, 361–375.

Tablan, V., Damljanovic, D., & Bontcheva, K. (2008b). A natural language query interface to structured information. *European Semantic Web Conference* (ESWC 2008), 361-375.

Vanessa, L., Victoria, U., Motta, E., & Michele, P. (2007). AquaLog: an ontology-driven question answering system for organizational semantic intranets. *Journal of Web Semantics*, *5*, 72–105.

Wang, H., Zhang, K., Liu, Q., Tran, T., & Yu, Y. (2008). Q2semantic: A lightweight keyword interface to semantic search. *European Semantic Web Conference* (ESWC '08), 584–598.

Wang, C., Xiong, M., Zhou, Q., & Yu, Y. (2007). PANTO: A Portable Natural Language Interface to Ontologies. *European Semantic Web Conference* (ESWC '07), 473-487.

Zhou, Q., Wang, C., Xiong, M., Wang, H., & Yu, Y. (2007). Spark: Adapting keyword query to semantic search. *International Semantic Web Conference (ISWC)/Asian Semantic Web Conference* (ASWC), 694-707.