PANTO: A Portable Natural Language Interface to Ontologies

Chong Wang, Miao Xiong, Qi Zhou, and Yong Yu

APEX Data and Knowledge Management Lab,
Department of Computer Science and Engineering,
Shanghai JiaoTong University, Shanghai, 200240, P.R. China
{wangchong,xiongmiao,jackson,yyu}@apex.sjtu.edu.cn

Abstract. Providing a natural language interface to ontologies will not only offer ordinary users the convenience of acquiring needed information from ontologies, but also expand the influence of ontologies and the semantic web consequently. This paper presents PANTO, a Portable nAtural laNguage inTerface to Ontologies, which accepts generic natural language queries and outputs SPARQL queries. Based on a special consideration on nominal phrases, it adopts a triple-based data model to interpret the parse trees output by an off-the-shelf parser. Complex modifications in natural language queries such as negations, superlative and comparative are investigated. The experiments have shown that PANTO provides state-of-the-art results.

1 Introduction

Ontology[1], which both explicitly represents the taxonomy of a domain (classes and properties) and stores a lot of knowledge (instances and instance relations), plays a key role in the semantic web by enabling knowledge sharing and exchanging [1]. However, in order to acquire the formal knowledge in ontologies, users have to be familiar with:

- the ontology syntax, such as RDF and OWL;
- some formal query language, such as RDQL[2] and SPARQL[3];
- the schema (structure) and vocabulary of the target ontology.

Consequently, as Bernstein et al. stated in [2], there is a *gap* between *the logic-based semantic web* and *real-world users*. In order to bridge the *gap*, this paper presents PANTO, a Portable nAtural laNguage inTerface to Ontologies, which offers users the convenience of acquiring needed information from formally defined ontologies. Specifically, users can express their information needs in natural language even without considering the syntax of RDF or OWL, the formal query language, or the schema and vocabulary of ontologies.

[1] In this paper, the term *ontology* refers to a knowledge base (KB) that includes concepts, relations, instances and instance relations that together model a domain.
[2] http://www.w3.org/Submission/RDQL/
[3] http://www.w3.org/TR/rdf-sparql-query/

E. Franconi, M. Kifer, and W. May (Eds.): ESWC 2007, LNCS 4519, pp. 473–487, 2007.

1.1 Background

Although the first natural language interface system came out more than three decades ago (LUNAR [3]), a fully portable and widely used system for formalized knowledge bases is still unavailable. As mentioned in [4], two major obstacles lie in the way:

Firstly, the ambiguity and complexity make it difficult for a machine to understand arbitrary natural language. The NLP community have been keeping on paying efforts in this area. The state-of-the-art statistical parsers [5] can reach about 90% in precision and recall. However, it is still well regarded as an "AI Complete" problem and far from totally resolved.

Secondly, even with correctly parsed natural language queries, a lot of challenges remain in translating them to correct formal queries:

- Vocabularies of the knowledge bases are controlled and lean compared with users', so it is a challenge to correctly map users' words to vocabularies of the knowledge bases.
- Together with the lexical and syntactic information of the parsed queries, semantic information in the knowledge bases can also be utilized to help formulate the formal query, but how to accomplish this is still an open problem.
- Different knowledge representations require different techniques to interpret the semantics of users' queries. For example, how to deal with queries containing negations in ontologies is different from that in databases. SPARQL has been recommended as the standard query language for the semantic web community, but few researches have been carried out to investigate the new opportunities and challenges in translating natural language queries into it.

1.2 Features and Contributions

PANTO focuses on the second obstacle mentioned above. It utilizes an off-the-shelf statistical parser StanfordParser [5] to deal with the first major obstacle. Multiple existing techniques and tools are integrated to interpret parse trees of natural language queries into SPARQL. The following are the major features and contributions:

Firstly, PANTO is designed to be ontology portable and no assumption is made about any specific knowledge domain. To help make sense of the words in the NL queries and map them to the entities (concepts, instances or relations) in the ontology, existing tools such as WordNet [6] and string metrics algorithms [7] are integrated.

Secondly, nominal phrase constituents in the parse trees are specially considered. We extract nominal phrases in the parse trees as pairs to form an intermediate representation called *QueryTriples*. Then, by utilizing knowledge in the ontology, PANTO maps *QueryTriples* to *OntoTriples* which are represented with entities in the ontology. Finally, together with *targets* and *modifiers* extracted from the parse trees, *OntoTriples* are interpreted as SPARQL.

Thirdly, we investigate certain problems in the process of translating natural language queries into SPARQL queries. The translation of advanced semantic features such as negation, comparative and superlative modifications are supported.

The rest of this paper is organized as follows. Section 2 introduces the system architecture. Section 3 describes how the ontologies and other resources are wrapped to construct a lexicon. Section 4 presents the key process to interpret parse trees into SPARQL queries. Section 5 elaborates on the experiments. Section 6 discusses about limitations and directs the future work. Section 7 describes related work and section 8 concludes this paper.

2 PANTO Architecture

Fig. 1 depicts the architecture of PANTO. It takes ontologies and natural language queries as input, and finally returns SPARQL queries as output. When an ontology is selected as the underlying knowledge base, the *Lexicon Builder*

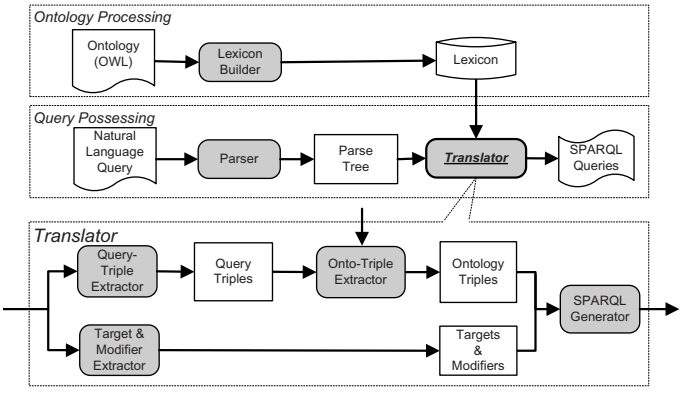

Fig. 1. Architecture of PANTO

automatically extracts entities out of the ontology to build the *Lexicon*. WordNet is utilized to find synonyms to enlarge the vocabulary of the ontology. Once the user inputs a natural language query, PANTO first invokes the *Parser* to produce the parse tree, which is then transfered to the *Translator*. The *Translator* is the core processing engine of PANTO. Upon receiving a parse tree, the *Translator* processes it in two ways in parallel:

- The parse tree is first transformed into *QueryTriples*, which are less restricted, for their predicates can be prepositions, verbs, phrases, or even omitted. Then *QueryTriples* are mapped to *OntoTriples* with the help of the *Lexicon*.
- The *Target and Modifier Extractor* extracts the potential words for *targets* (variables after "SELECT") and *modifiers* (information related to "FILTER", "UNION", etc.) of the target SPARQL queries from the parse tree.

Finally the domain-compliant *OntoTriples*, *targets* and *modifiers* are sent to the *SPARQL Generator* to produce SPARQL queries.

3 The Lexicon

The *Lexicon* is mainly designed for making sense of words in natural language queries and mapping them to entities in the ontology. It is composed of the following contents:

Ontology Entities. This is the most important part of the *Lexicon*. Entities in the ontology, including classes (concepts), properties (relations), and instances (individuals), are extracted and stored for fast access and matching. Since different ontology entities may have the same name (e.g. "Mississippi" river and "Mississippi" state) and one ontology entity may have different names (e.g. "US", "United States", and "USA" denote the same entity "United States of America" in the ontology), ontology entities and their names are put into a special hash table, in which a key maps to a set of ontology entities and an ontology entity can be obtained by different keys. Given a word from the natural language query, the *Lexicon* will acquire a set of possible entities. Proper nouns (e.g. "New Mexico") are also extracted from the ontology for fast access and matching.

General Dictionaries. In order to help bridge the *gap* between user vocabulary and ontology vocabulary, general dictionary WordNet is utilized. The synsets in the dictionary defined by linguists enable PANTO to match "work" in user's query to concept "Job" in the ontology. Also with the help of the dictionaries, we are able to retrieve the property "length" when the user asks "how *long* ...". This module is open and other thesauri can also be adopted if available.

User-Defined Synonyms. Since users may use jargons and abbreviations to denote entities, words from general dictionaries only may not be enough. Therefore, the *Lexicon* allows users to define their own "synonymy words" (a set of words that match the same entity in the ontology). This will be helpful when PANTO is adopted to a certain domain.

Note that, the user-defined synonyms are not mandatory for the *Lexicon*, and all the mandatory contents are extracted in a totally automatic way. Therefore, the construction of the *Lexicon* is portable.

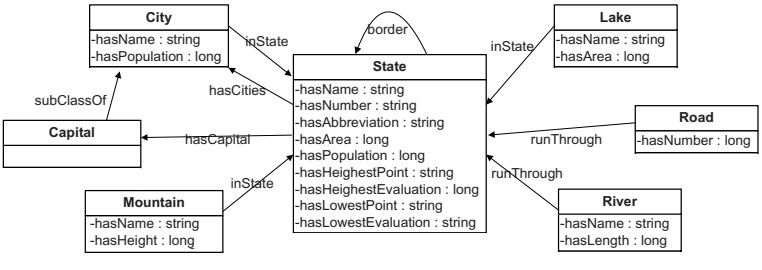

Fig. 2. The schema definition of *geography* ontology (the following examples are based on this ontology)

4 Translator: Translating Parse Tree to SPARQL

The translator is the backbone of PANTO. Our basic idea for translating natural language queries into formal ones is to specially consider nominal phrases. We observe that a natural language query can usually be viewed as the combination of multiple nominal-phrase pairs. From the parse trees by a deep parser, such pairs are easily recognized. Inside each pair is a verb phrase, a preposition, a conjunction or the like to represent the relationship between the two nominal phrases. At the same time, nominal phrases or words also play an important role in ontologies which store facts to model a domain. The facts are explicitly or implicitly stated in the triple form <*subject*, *predicate*, *object*>. The *subject* and the *object* may be classes, instances or literal values and usually should be named with nominal words or phrases [8]. The *predicate* may be prepositions, verbs, verb phrases and so on, and sometimes may also be nominal phrases. Because a nominal-phrase pair represents some kind of semantic relationship between the two nominal phrases, we expect it to be mapped to a triple in the ontology. Fig. 2 presents the schema definition of an example ontology and Fig. 3 depicts the processing steps of an NL query to this ontology. In the following subsections, we will detail the whole process of translating a parse tree of a natural language query into SPARQL based on the above idea.

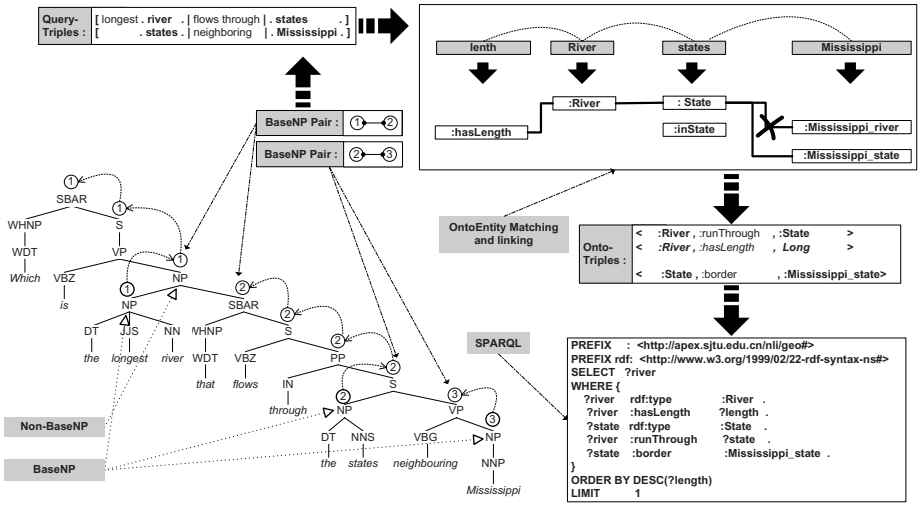

Fig. 3. Parse tree of query "which is the longest river that flows through the states neighboring Mississippi" by StanfordParser (with the default root node "ROOT" removed for brevity). *BaseNP*, *Head BaseNP* propogation, *BaseNP pairs*, *QueryTriples*, *OntoEntity* matching, linking *OntoEntities* as *OntoTriples*, *OntoTriples* and final SPARQL query are illustrated.

4.1 QueryTriple Extractor

In this part, we extract the nominal-phrase pairs out of the parse tree to form the intermediate representation *QueryTriples*. This includes the following steps:

Identify and propagate Head BaseNP. In the parse tree, a nominal phrase is tagged as NP[4]. Since NPs may be nested, we need to find those that can be mapped to entities in the ontology. We introduce the notion *BaseNP* (Fig. 3). **BaseNP** is formally defined as *NonrecursiveNP* (and *BaseNP* is called for short) in [9], where it refers to an NP that does not directly dominate an NP itself, unless the dominated one is a possessive NP (i.e. it directly dominates a POS-tag "POS"). Here we slightly extend the concept of *BaseNP* to also include a constituent (some WHNP) if it contains nouns but does not dominate an NP. We make this extension to capture all nominal words, which may not be parsed to be contained in a *BaseNP* by a statistical parser. For example, StanfordParser parses "how many rivers" in the query "how many rivers does alaska have" as (*WHNP*(*WHADJP*((*WRB how*)(*JJ many*))(*NNS rivers*)). There is a noun "rivers" inside WHNP, but there is no NP.

With this definition, we first identify all the *BaseNPs* from a parse tree. After that, we prepare for identifying and linking those related NPs as pairs. We follow the rules used by Michael Collins [9] for mapping from trees to dependency structures (StanfordParser also uses these rules and this is one of the reasons why it is utilized in PANTO). The difference is: Collins used those rules to find *head words* (see [9] for details), and we use a subset of the rules to propagate *head BaseNPs*. The only modification to the rules is: if there is a node containing conjunctive children nodes, the *head BaseNPs* of all the children will be

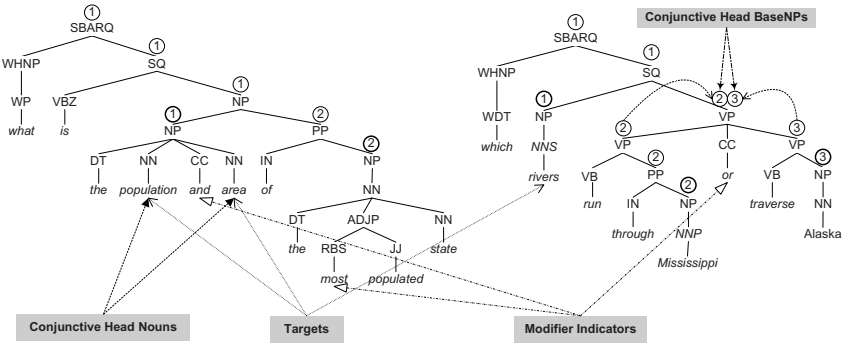

Fig. 4. Parse trees of the queries "what is the population and area of the most populated state" and "which rivers run through Mississippi or traverse Alaska" by StanfordParser (with default root node "ROOT" removed for brevity). *Targets*, *modifier indicators*, conjunctive *Head Nouns* and conjunctive *Head BaseNPs* are illustrated.

[4] NP, short for "Nominal Phrase", is a syntactic tag in parse tree. In the following, we will use such syntactic tags as well as POS tags without explanations.

propagated to it as its *head BaseNPs* (Fig. 4). All these operations can be done via a bottom-up traversal along the parse tree.

Link BaseNP Pair to form QueryTriple. After the basic constituent *BaseNPs* are identified and propagated on the parse tree, we link them one another where there is modification relationship to form *BaseNP pairs*. The process is: *for each BaseNP, traverse up towards the root node and link it with the first different head BaseNP(s) as BaseNP pair(s)*. The two *BaseNPs* in such a pair, together with the words which syntactically connect them, form a *QueryTriple* (Fig. 3). A **QueryTriple** is a triple in the form of [*subject* |*predicate* |*object*]. The *subject* and the *object* are the two *BaseNPs* and the modifiers to them extracted from the original query. The *predicate* is composed of the words (may be a preposition, a verb phrase, etc) that connect the *subject* and the *object* together.

Specify internal structure for QueryTriple. A linked *BaseNP* pair is only a raw *QueryTriple*. Since a *BaseNP* may contain more than nominal words, we need to separate different contents. First of all, the *head nouns* for the *BaseNPs* are identified. A **Head Noun** is a nominal word acting as *head word* [9] in a *BaseNP*. The rules to find *head noun* again follows [9], with the exception that when conjunction exists, treat all nominal words of conjunctive relations as *head nouns* (Fig. 4). As the next step, the internal structure of *QueryTriples* are specified. For the *subject* and the *object* of each *QueryTriple*, the internal structure is in the form of [*pre-modifier . head noun . post-modifier*]. Here, only the *head noun* is mandatory and the *pre/post-modifiers* represent the words modifying the *head noun* or the whole *BaseNP*.

4.2 OntoTriple Extractor

QueryTriples are only the intermediate forms of the user's query. We need to map them to the semantic content in the ontology. This is carried out as below. First, the *OntoTriple Extractor* matches the words (especially nominal words) with the *OntoEntities* (**OntoEntity**, short for *Ontology Entity*, represents concepts, instances, relations/properties in the ontology). Then, it makes use of lexical and syntactic information extracted with *QueryTriples* to find the semantic relationships among *OntoEntities*, which will be explicitly represented with *OntoTriples*. The detail is as follows:

Map user words to OntoEntities. The first is to find the corresponding *OntoEntities* for each word in the query. For each *QueryTriple*, we retrieve the matching *OntoEntities* for the *head nouns* of the *subject* and the *object*, by invoking the *Lexicon*. The *Lexicon* employs a number of matching methods, which can be classified as two types: (1) *semantic matching* mainly uses general dictionaries like WordNet to find synonyms of words; (2) *morphological matching* uses WordNet, string metrics or heuristic rules (e.g. algorithms to find abbreviations, which may also be separately designed when PANTO is adopted to a particular domain) to find morphologically similar matchings. Different matching methods are sometimes combined to find matching entities for a word. For example, the word "states" gets the matching list {State, inState} (Fig. 3). "State" is a *class*

entity retrieved by morphological matching and "inState" is a *property* entity matched by a heuristic rule based on naming conventions for ontology and string metrics algorithms.

Map QueryTriples to OntoTriples. An **OntoTriple** (short for *Ontology Triple*) is a triple that is compatible with some statements in the ontology, in the form of <*subject*, *predicate*, *object*>, where the *predicate* can be a relation (property) and the *subject* or the *object* can be a concept (class) or an instance. When the *predicate* is a *datatype* property, the *object* must be a literal value or the value type. A nominal word may also be mapped to the *predicate*, besides the *subject* and the *object*. Therefore, the *subject* or the *object* in the *QueryTriple* does not necessarily be mapped to that in the *OntoTriple*. In PANTO, 11 cases are enumerated for OWL ontology to generate *OntoTriple*(s) from two *OntoEntities*. For example, if one *OntoEntity* is a *property* and the other is a *class* which is the domain of the *property*, one *OntoTriple* is generated; if both are *properties* and can be related by a *class*, two *OntoTriples* are generated. With different combinations of the matching *OntoEntities* of the *head nouns* in the *subject* and the *object* of the *QueryTriple*, multiple *OntoTriples* will be generated. When the *predicate* of a *QueryTriple* is not empty, we use it to verify the generated *OntoTriples*. For example, from the *QueryTriple* [river | flows through | mississippi], we get two *OntoTriples* <:Mississippi, rdf:type, :River> and <:River, :runThrough, :Mississippi>. Since "flows through" can be mapped to ontology property ":runThrough", we discard the first *OntoTriple*. On the other hand, if the two *OntoTriples* are generated from *QueryTriple* [river | | mississippi], we discard the second one. Because this pattern is usually used by people to indicate an entity [10], in such a case we believe with high confidence that "mississippi" should be mapped to the entity "Mississippi_river" rather than "Mississippi_state". Thus we remove all the other matching *OntoEntities* for the word "mississippi", and finally also discard the triple <:Mississippi, rdf:type, :River> and only hold that *OntoEntity*. Besides the *OntoTriples* generated above, this module also generates *OntoTriples* from inside a *BaseNP*. Take the *BaseNP* "the longest river" in Fig. 3 as an example, the *OntoTriple Extractor*, with the help of WordNet, transforms "longest" to "long" and then to the nominal form "length". Now "length" is used to match *OntoEntities*. These matching *OntoEntities* are then used to generate *OntoTriples* with those matching *OntoEntities* of "river". Finally, a valid *OntoTriple* <:River, hasLength, Long> is generated.

Link OntoTriples. A natural language query represents the semantic relationships and constraints among different concepts and individuals in the domain. When multiple *BaseNP pairs* are available, there are *BaseNPs* shared by two or more *QueryTriples*. Therefore, a valid *OntoTriple* set should be linked one another to form a tree (Fig. 3). Since one word may match multiple *OntoEnties*, there may be different combinations and multiple valid *OntoTriple* result sets.

4.3 Target and Modifier Extractor

To translate a natural language query into a SPARQL query, we must find the *targets*, i.e. the words that correspond to the variables after "SELECT" in

the resultant SPARQL query. This process is as follows: first find the allowed wh-word [10] like "what", "who" and "how" or an enumerated set of command words like "list", "give me" and "return"; then take the nouns in the same or the directly followed constituent as *targets* (Fig. 4). Detailed rules vary for different question/command words, and are usually common for different domains.

Filter is provided in SPARQL to enable the users to specify constraints on relations and variables. *Solution Modifier* is provided for the users to carry out operations (*Order by*, *Limit*, *Offset*) on the result. In natural language queries, both of these are expressed through certain types of words, which we call *modifier indicators* (Fig. 4). In the current version of PANTO, we mainly deal with the following: (1)*negation*, including "not" and "no"; (2)*superlative*, superlative words will be tagged as "JJS" (superlative adjective) or "RBS" (superlative adverb); (3)*comparative*, comparative words will be tagged as "JJR" (comparative adjective) or "RBR" (comparative adverb); (4)*conjunctive/disjunctive*, including "and" and "or".

The extractor records the positions and types of *targets* and *modifier indicators*, and then send them as input to the SPARQL Generator.

4.4 SPARQL Generator

The *targets* and *modifiers* are extracted from the parse tree, and it is straightforward to relate them with corresponding *OntoEntities* and *OntoTriples* extracted by the *OntoTriple Extractor*. After that, we interpret them into SPARQL:

SELECT. *OntoEntities* matched with *target* words are interpreted as variables after "SELECT" according to the following rules: (1) If it is a *class* entity ":SomeClass", interpret it as the variable "?someclass", and add a *triple pattern*[5] <?someclass, rdf:type, :SomeClass> to the "WHERE" clause. (2) If it is an *RDF Literal Type* entity, such as "Long", "String" or the like, directly interpret it as a variable, e.g. "?long", "?string", etc.

WHERE. An ordinary *OntoTriple* is directly interpreted as a *triple pattern* after "WHERE". *Instance* and *property* entities are directly interpreted as their URIs in the ontology. As for *class* entities or *RDF Literal Type* entities in each *OntoTriple* (except those with the *property* "rdf:type"), interpret them as variables according to the above rules for *targets*. For some complex queries, there may be multiple words mapped to the same *class* entity. For example, the two "state" in the query "what state borders the state that has the largest population?" are both mapped to the *class* entity ":State". However, they should obviously be interpreted as different variables. Our current solution is to always interpret such entities as different variables. [11] mentioned a technique to check whether such two words mean the same or not, by comparing their local contexts in the query according to some heuristics, but it requires ad hoc rules. According to the current experiments, our approach works fine. More investigations will be carried out as future work. *RDF Literal Type* entities are treated similarly.

[5] http://www.w3.org/TR/rdf-sparql-query/

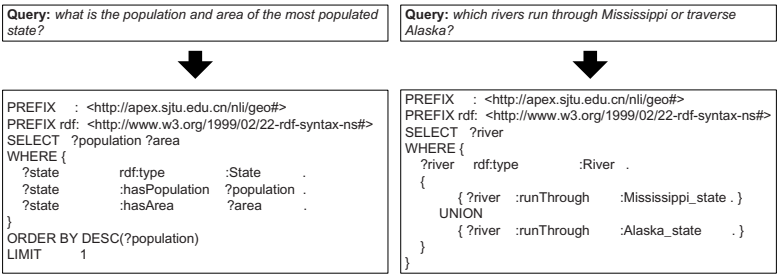

Fig. 5. SPARQL queries for the two example queries in Fig. 4

FILTER and Modifiers. Corresponding to the *modifier indicators* mentioned above, this part currently includes the following: (1)*negation.* Negations are interpreted by utilizing the operators "!" and "bound" of SPARQL. For example, "not" in the query "which river's length is not 5000" is interpreted as a FILTER clause "(FILTER(?length != 5000))". Negation on *property* (relation) are interpreted by a combination of "OPTIONAL" and "FILTER". Take the query "which rivers do not run through Alaska" as an example, "do not" is interpreted as "{OPTIONAL {{?river :runThrough ?state.} FILTER (?state = :Alaska)} FILTER (!bound(?state))}". However, some kinds of negations can not be interpreted, e.g. queries like "which rivers do not run through states bordering Alaska". (2)*superlative.* Superlative modification on *datatype* can be interpreted with *Order By* and *Limit* (*Limit* can be removed if there are multiple results) of SPARQL (e.g. the NL query and its resultant SPARQL query shown in Fig. 3), but superlative modifications that require the functionality of *count* are not supported (e.g. "most" in query "which river runs through the most states" can not be expressed with the current version of SPARQL). (3)*comparative.* Similar as above, currently only comparative modification on *datatype* is supported. (4)*conjunctive/disjunctive. OntoTriples* related to conjunctive constituents (conjunctive *BaseNPs* or conjunctive *head nouns*) indicated by "and" are interpreted as conjunctive *triple patterns* to the "WHERE" clause by default. Since "and" may sometimes be used in disjunctive relation (e.g. in the query "list all the cities in California and Maine"), we need to link their *triple patterns* with a "UNION" in such a situation. Currently, when the two conjunctive words are both mapped to the same *class* entity (or *instances* of the same *class*), multiple SPARQL queries are produced for different interpretations of "and". Those *OntoTriples* related to "or" are always interpreted with a linking "UNION".

As an example, Fig. 3 depicts the final SPARQL query. Fig. 5 depicts the final SPARQL queries for the two example queries used in Fig. 4.

5 Experiments and Evaluation

The goal of the experiments is to quantitatively assess the performance and effectiveness of our approach in the current implementation.

Table 1. Performance of PANTO. Row 2 shows the number of original Mooney queries and row 3 shows that of the selected testing queries (duplicated ones are removed).

Domain	*Geography*	*Restaurant*	*Job*
Original Mooney Queries#	880	250	641
Selected Testing Queries#	877	238	517
Precision	88.05%	90.87%	86.12%
Recall	85.86%	96.64%	89.17%

5.1 System Implementation and Experiment Setup

PANTO was implemented as a stand-alone web application. In the current version, it adopts Protégé API[6] to access the underlying ontologies. The version of WordNet used in the system is 2.1. The lexicalized StanfordParser[7], version 1.5.1 without additional training, is adopted as the statistical parser. It produces one parse tree for each input. The experiments were performed on a PC with 2.4GHz P4 CPU and 1G Memory.

Test Data. The test data are based on those provided by Mooney[8] which have been widely used to evaluate natural language interfaces [2,12,13]. There are test data and queries on three domains: one about geography data in the United States, one about restaurant information and the third about job announcements. We translated the original three Prolog databases into OWL as the testing ontologies. There are several hundreds of natural language queries for each domain. We removed the duplicated ones in each query set, Table 1 presents the numbers of queries for experiments.

5.2 Results and Discussion

Performance. With the initialization time (i.e. loading ontologies and parsers) excluded, the total average processing time of a query was less than one second (0.787s) and the running time is related with the scale of the ontology and the complexity of the query (the length and the number of clauses).

Correctness. In order to assess the correct rate that how many of the translated queries correctly represent the semantics of the original natural language queries, we compare the output with the manually generated SPARQL queries. The metrics we used are *precision* and *recall*. For each domain, *precision* means the percentage of correctly translated queries in the queries that PANTO produced an output; *recall* refers to the percentage of queries that PANTO produced an output in the total testing query set. Since the queries are originally prepared for evaluating natural language interface to database, some features in the queries are not supported by the current version of SPARQL, but we also count them as correct if they match the manually generated ones.

[6] http://protege.stanford.edu/download/registered.html
[7] http://nlp.stanford.edu/software/lex-parser.shtml
[8] http://www.cs.utexas.edu/users/ml/nldata.html

To the best of our knowledge, the only existing system that also translates generic natural language queries into SPARQL is Querix [14], which provides preliminary results on the geography dataset with 215 selected queries. Querix claimed to achieve a precision of 86.08% and recall 87.11%. From Table 1 we can see that PANTO provides comparable results.

Coverage. In this part we analyze the effectiveness of the approach to interpreting the queries in a triple-based model. The experiments on the Mooney data and queries show that all the correctly parsed natural language queries can be correctly translated into *QueryTriples* and then be mapped to *OntoTriples* at a high accuracy. What's more, we also parsed and analyzed the 170 sample queries presented on the AquaLog web site[9]. AquaLog claims to be able to parse these kinds of queries, and with PANTO, all of them can generate pretty good *QueryTriples*. Hence we can expect that the PANTO approach can cover the query scope supported by AquaLog.

6 Limitations and Future Work

The limitations with the current version of PANTO mainly include the following:

Restrictions on Query Scope. Though the triple-based analysis is effective to cover a broad range of natural language queries, the supported query scope is still limited. First, PANTO depends on an off-the-shelf parser to correctly parse the NL queries and thus is limited by the NLP techniques. But we can continually utilize the new achievements in NLP community. Second, it can not totally interpret semantics that is beyond the expressiveness of SPARQL (e.g. queries involving *count* on instances). When more features are added to SPARQL, these limitations are expected to be resolved.

Weakness in User Interaction. At present, PANTO is not a full-fledged system. It focuses on the query processing step and is currently weak in supporting complex user interactions. However, a well-designed interaction model can enable users to paraphrase the query and guide them correctly express their information need with system processable input. In future, we will investigate more on user interaction to make PANTO more effective.

Scalability. The ontologies used for evaluation is relatively small and all processing operations are carried out in memory. An investigation on the system performance with larger ontologies will be part of the future work. Database and indexing techniques will also be involved.

7 Related Work

Natural language interface to knowledge bases, which can help ordinary users express their information needs in natural language that they are familiar with and

[9] http://kmi.open.ac.uk/technologies/aqualog/examples.html

can consequently populate the knowledge bases, has been studied for decades [15, 16]. According to the ability of processing the natural language input, such natural language interfaces can be classified into two categories, namely, full natural language interface and restricted natural language interface.

Masque [17] is a typical natural langue interface to databases. It first transforms the natural language query into an intermediate logic representation and then translates the logic query into SQL. Systems which employ machine learning methods for transforming natural language query into formal logic representation or formal query have been studied for many years too. With the training on domain specific sentences, these systems [18,19,20] gain a good result (precision can be higher than 95% and recall can reach 80%). However, they need a lot of domain specific training. In order to avoid the defects brought up by NLP parsers, many systems only use the shallow parsing result of NLP tools, e.g. POS tags, chunks, etc. Rodrigo et al. tried to break down the query into words and form a formal query with words in their semantic search engine for the international relation sector [21]. Kang et al. have also formed the SQL query with keywords in [22]. As it is summarized by the authors in [21], "...the query construction is at present the weakest link in the chain...". Since PANTO combines the *OntoTriples* and the modifiers generated from the parse tree and the *Lexicon*, it can easily constructs the SPARQL query. NaLIX [4,11] is a generic natural language interface to XML Database. It focuses on correct parse trees output by a dependency parser and translated them into XQuery. Querix [14] is a domain-independent natural language query interface to Ontologies based on clarification dialogs. It is the only known system that also translates full natural language queries into SPARQL. Similar to PANTO, Querix also adopts an off-the-shelf parser (also StanfordParser), but it differs from PANTO in analyzing the parse trees. Querix directly uses the POS tags and a set of heuristic rules to extract skeletons (e.g. Q-V-N for "what are the population sizes"), while PANTO identifies *BaseNPs* and utilizes structure information inside and among the *BaseNPs*.

Because of the complexity and ambiguity of full natural language, many systems only accept queries which are in a subset of natural language. Controlled natural language, which restricts the terminology and grammar and is equivalent to First Order Logic [23], avoids the ambiguity of full natural language. Bernstein et al. [2] have adopted Attempto Controlled English to query ontologies and Nelken et al. [24] have employed controlled language to query historical databases. The queries are in the form of natural language, but users have to first learn the syntactic restrictions to make sure they are in the "controlled" set. PRECISE [13] defines a notion of "semantic tractable" questions on database and can translate them into SQL queries. However, all tokens must be distinct and questions with unknown words are not semantically tractable and cannot be handled. In contract, with the *Lexicon*, PANTO can deal with questions even some of them are not "semantic tractable". Pattern based methods are also widely used in natural language interfaces. Lopez et al. [25] have classified questions into 23 categories in AquaLog. If the input query is classified into some

category, AquaLog will process it correctly. However, due to the limited coverage of the patterns, many queries will be left unresolved. Comparing to other systems, AquaLog is more similar to PANTO. Its underlying knowledge base is ontology, it also adopts a triple-based intermediate presentations and it is the one that introduces the notion of *query-triple* and *onto-triple*. The difference is, AquaLog is based on a shallow parser and depends on handcrafted grammars to identify *terms*, *relations* for composing *query-triples*, while the parse tree by the deep parser provides PANTO more modification information between nominal phrases. What's more, different from the *query-triple* of Aqualog, our *Query-Triple* are always formed with two nominal phrases (the *BaseNP pair*). Bernstein et al. have also proposed a guided natural language search engine [12,26] to help users form the query and avoid ambiguity, but the processing ability of the system is also limited to the defined grammar.

8 Conclusion

This paper presents PANTO, a portable natural language interface to ontologies. Based on the observation that nominal words or phrases play an important role in both natural language query and ontology triples, PANTO adopts a triple-based data model as the intermediate representation to translate natural language queries into SPARQL. The experiments on three different ontologies have shown that the PANTO approach produces promising results. Our approach helps bridge the *gap* between *the logic-based semantic web* and *real-world users*.

Acknowledgments

This work is carried out as part of a university joint research project between IBM China Research Lab and Department of Computer Science and Engineering, Shanghai JiaoTong University. The authors sincerely thank the anonymous reviewers for their valuable comments. We also would like to thank Zhangmei Yao for his participation in this research.

References

1. Chandrasekaran, B., Josephson, J.R., Benjamins, V.R.: What Are Ontologies, and Why Do We Need Them? IEEE Intelligent Systems **14**(1) (1999) 20–26
2. Bernstein, A., Kaufmann, E., Göhring, A., Kiefer, C.: Querying Ontologies: A Controlled English Interface for End-Users. In: International Semantic Web Conference. (2005) 112–126
3. Woods, W., Kaplan, R., Webber, B.: The Lunar Sciences Natural Language Information System: Final Report. Technical report, Bolt Beranek and Newman Inc., Cambridge, Massachusetts (1972)
4. Li, Y., Yang, H., Jagadish, H.V.: NaLIX: an interactive natural language interface for querying XML. In: SIGMOD Conference. (2005) 900–902

5. Klein, D., Manning, C.D.: Accurate Unlexicalized Parsing. In: ACL. (2003) 423–430
6. Fellbaum, C. In: Wordnet: An Electronic Lexical Database. Cambridge: MIT Press (1998)
7. Cohen, W.W., Ravikumar, P., Fienberg, S.E.: A Comparison of String Distance Metrics for Name-Matching Tasks. In: IIWeb. (2003) 73–78
8. Noy, N.F., McGuinness, D.L.: Ontology Development 101: A Guide to Creating Your First Ontology. Technical Report SMI-2001-0880, Stanford University School of Medicine (2001)
9. Collins, M.: Head-driven Statistical Models for Natural Language Parsing. PhD thesis, University of Pennsylvania (1999)
10. Quirk, R., et al. In: A Comphrehensive Grammar of the English Language. Longman, London (1985)
11. Li, Y., Yang, H., Jagadish, H.V.: Constructing a Generic Natural Language Interface for an XML Database. In: EDBT. (2006) 737–754
12. Bernstein, A., Kaufmann, E., Kaiser, C.: Querying the Semantic Web with Ginseng: A Guided Input Natural Language Search Engine. In: 15th Workshop on Information Technology and Systems (WITS 2005). (2005) 45–50
13. Popescu, A.M., Etzioni, O., Kautz, H.A.: Towards a Theory of Natural Language Interfaces to Databases. In: Intelligent User Interfaces. (2003) 149–157
14. Kaufmann, E., Bernstein, A., Zumstein, R.: Querix: A Natural Language Interface to Query Ontologies Based on Clarification Dialogs. In: 5th International Semantic Web Conference (ISWC 2006), Springer (2006) 980–981
15. Androutsopoulos, I., Ritchie, G., Thanisch, P.: Natural Language Interfaces to Databases - An Introduction. Natural Language Engineering **1**(1) (1995) 29–81
16. Copestake, A., Jones, K.S.: Natural Language Interfaces to Databases. Knowledge Engineering Review **5**(4) (1990) 225–249
17. Androutsopoulos, I., Ritchie, G., Thanisch, P.: An Efficient and Portable Natural Language Query Interface for Relational Databases. In: 6th International Conference on Industrial and Engineering Applications of Artificial Intelligence and Expert Systems. (1993) 327–330
18. Zelle, J.M., Mooney, R.J.: Learning to Parse Database Queries Using Inductive Logic Programming. In: AAAI/IAAI, Vol. 2. (1996) 1050–1055
19. Thompson, C.A., Mooney, R.J.: Automatic Construction of Semantic Lexicons for Learning Natural Language Interfaces. In: AAAI/IAAI. (1999) 487–493
20. Zhang, L., Yu, Y.: Learning to Generate CGs from Domain Specific Sentences. In: ICCS. (2001) 44–57
21. Rodrigo, L., Benjamins, V.R., Contreras, J., Patón, D., Navarro, D., Salla, R., Blázquez, M., Tena, P., Martos, I.: A Semantic Search Engine for the International Relation Sector. In: International Semantic Web Conference. (2005) 1002–1015
22. Kang, I.S., Na, S.H., Lee, J.H., Yang, G.: Lightweight Natural Language Database Interfaces. In: NLDB. (2004) 76–88
23. Fuchs, N.E., Schwertel, U., Torge, S.: Controlled Natural Language Can Replace First-Order Logic. In: ASE. (1999) 295–298
24. Nelken, R., Francez, N.: Querying Temporal Databases Using Controlled Natural Language. In: COLING. (2000) 1076–1080
25. Lopez, V., Pasin, M., Motta, E.: AquaLog: An Ontology-Portable Question Answering System for the Semantic Web. In: ESWC. (2005) 546–562
26. Bernstein, A., Kaufmann, E., Kaiser, C., Kiefer, C.: Ginseng: A Guided Input Natural Language Search Engine for Querying Ontologies. In: 2006 Jena User Conference. (2006)