A Natural Language Query Interface to Structured Information

Danica Damljanovic, Valentin Tablan, and Kalina Boncheva

Department of Computer Science
University of Sheffield
Regent Court, 211 Portobello Street
S1 4DP, Sheffield, UK
{d.damljanovic,v.tablan,k.boncheva}@dcs.shef.ac.uk

Abstract. Extracting knowledge from semantic markup is essential once that knowledge is in structured form and semantically enriched. Existing formal languages for querying the knowledge bases are not convenient for non-expert users. Developing more user-friendly interface to the structured data would enable non-experts to query such a knowledge without prior training, while for experts it will significantly ease the process of querying. Using Natural Language Interfaces (NLI) is the most natural way to achieve user-friendliness of such interfaces. As natural human language is known for its ambiguity and complexity there is need to narrow its usage by using controlled languages (CL). On one side CL limits the power of expressiveness, but on the other provides simplicity and possibility to retrieve data without any previous learning or training. We present Controlled Language for Ontology Editing and Querying (CLOnE-QL) - a tool for querying the knowledge represented by ontologies using text-based human understandable language. CLOnE QL has a very simple interface, requires no training and can be easily embedded in any system or used with any ontology or knowledge base.

Key words: ontology, ontology-aware annotations, semantic search, natural language queries, natural language interfaces

1 Introduction

Along with formal languages for representing ontologies (e.g., OWL, RDF), many languages for querying them are also available (e.g., SPARQL, SeRQL). These languages are usually complex and each has specific syntax. For query construction and understanding of the results users must understand not only the language itself, but also ontologies and ontology languages. Even for an expert in this field, writing queries is error-prone task whilst the syntax is not so easy to learn.

Developing user-friendly interfaces to structured data such as that in ontologies would be a potential way to ease the process of creating queries using formal expressions for expert users, whereas for non-experts it would be a new opportunity to query the knowledge base without any background knowledge.

Up to date, many interfaces to knowledge bases are already developed. Some of them provide graphical interface where users can browse ontology, and also extract some knowledge from it by executing queries expressed in formal languages such as SPARQL. The other are more user-friendly and provide forms for performing semantic search based on underlying ontology whilst hiding the complexity of formal languages. The most sophisticated ones provide only one text box for a query, and require no training of the user, accepting Natural Language (NL) queries as an input.

According to the interface evaluation conducted in [1], systems developed to support NL interfaces seems like the most acceptable by end-users. [1] tested usability of Semantic Webs capabilities to the general public with 48 users and four types of query language interfaces. The queries were executed against the OWL knowledge bases. This study showed that the full-sentence query option was significantly preferred to keywords, a menu-guided, and a graphical query language interfaces.

Despite user preferences, the development of accurate NLI systems is 'a very complex and time-consuming task that requires extraordinary design and implementation efforts' [1]. Namely, the NLI systems vary from those that are domain independent, portable and with low performance, to more domain specific ones being portable only prior to customisation, and with a very high performance. However, the customisation of the latter is usually very expensive as it is performed by humans, often being experts in one field or another. The key for bridging the gap between the two extreme would be automatization of the process of domain specific knowledge creation. The nature of semantic markup have power to give this sophisticated dimension to NLIs by arising the possibility to extract domain-specific knowledge from the formal structures automatically, resulting in creation of domain-independent NLI with a high performance. However, the quality of semantic markup in formal structures have to be very high. The question is what does sufficient quality for the semantic markup means and how to achieve this? Experts for customising NLI systems usually have to create domain specific knowledge so that that knowledge can be used for question answering. To create this knowledge they usually have to add descriptions or labels to the relevant terms or to map one terms with the others. If an ontology would be created so that each concept and relation is accompanied with human understandable label or description, the automatization of the domain specific knowledge creation would be feasible.

The other problem with NLI systems is hidden in the nature of human languages, which are being well-known for their ambiguity and complexity. To overcome this problem it would be feasible to limit the usage of NL by introducing a controlled language. A controlled language (CL) is a subset of a natural language that includes certain vocabulary and grammar rules that have to be followed. On one side CL limits the power of expressiveness, whilst on the other provides simplicity and possibility to retrieve data without any previous learning or training.

We present CLOnE QL (Controlled Language for Ontology Editing and Querying) - portable, domain-independent and simple NLI tool that accepts text-base queries as an input and transforms it to the formal language query, such as that of SeRQL (see figure 1). The queries are then being executed against the given ontology and/or knowledge base usually containing instances of the concepts from the ontology and relations between them.

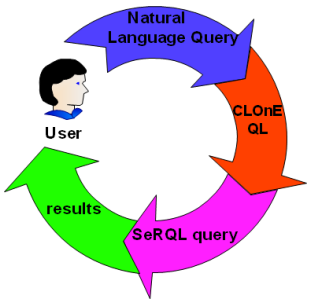

Fig. 1. Process of creating formal language queries (e.g., SeRQL) from human language

CLOnE QL was inspired by CLOnE (Controlled Language for Ontology Editing) [2]. CLOnE provides users to edit ontologies using natural language. Similarly, CLOnE QL enables users to query the existing knowledge by transforming input NL queries into SeRQL queries using GATE resources for natural language processing.

2 Background

One of the ways to search the knowledge base and/or ontology is using existing tools that enable browsing and navigation. One of the most popular is Protege [3]. Expert users would most probably understand most of the defined concepts and relations between them while browsing an ontology in Protege editor, but for browsing a huge amount of data, they might get lost among hundreds of tree nodes. However, for such cases it's possible to use Protege Query Interface, where one can specify the query by selecting some options from given list of concepts and relations. Alternatively, for even more expressive search users can use a simple interface where they type in a query using a formal language such as SPARQL and get the results. Such a tool is useful for experts that are familiar with query languages, specifically SPARQL in this case, although they also have to be experienced Protege users.

One step towards more user-friendly interface is a semantic search form of some knowledge management platforms such as KIM [4]. Their interface enables querying knowledge bases by either instantiating a query from a set of given templates, or by constructing a SeRQL query using a form-based interface. Consequently, users are either restricted in what they can search for, or they need to

be familiar with the query language and the underlying ontology. CLOnE QL, as well as other NLI systems, differ from KIM in that it saves users not only from the complexity of the query language(s), but also from the necessity to be familiar with ontologies and semantic search. Moreover, CLOnE QL has no limitations in the number of concepts to be included in a query, whereas in KIM the interface will not allow to construct queries comprising the combination of more than 3 concepts. Using NLIs provides flexibility in a manner that there is no limitation of this kind: it is the user who decide how many concepts will be included in the query.

The most simple interface usually used by NLI systems is that of search engine's such as Google's. [5] present SemSearch - a concept-based system which claims to have Google-like Query Interface. It requires a list of concepts (classes or instances) as an input query separated by colon (e.g., 'news:PhD Students' is a query that results in all instances of class News that are in relation with PhD Students). The idea of having a simple form for semantic search queries is very good. On the other side, SemSearch does not consider properties, and neither does it disambiguate in cases when there is more than one relation between the two concepts. It also does not accept natural language questions such as 'how many PhD Students are registered at the University of Sheffield?'. Retrieving relevant properties is of great importance in our approach, resulting in SeRQL queries that are of a high accuracy.

AquaLog [6] is perhaps the system closest to ours, as it also uses a controlled language for querying ontologies. It is also coupled with a learning mechanism, so that its performance improves over time, in response to the vocabulary used by the end users. However, this system heavily relies on language processing tools and requires syntactically correct sentences as an input. Our approach differs in that of being much more robust with respect to mistakes in the controlled language and, in addition to question-based queries, it also supports concept-based ones such as 'hotels London' instead of 'list hotels located in London'. Additional limitation of Aqualog is that of being capable of dealing with questions containing a maximum of two triples only. CLOnE QL has no limitations of this type, as it would give results for any number of concepts appearing in a query, meaning that the maximum of the triples included in one query is not limited.

Additionally, Aqualog requires configuring and adding so called 'pretty names' to the ontology resources, whereas in CLOnE QL we derive everything from the ontology - we believe that ontologies, even primarily built to enhance the structure of data and make it understandable to machines - also have to be understandable to humans, wherefor ontology designers should follow recommended standards i.e. by giving 'pretty names' to the values of properties such as rdfs:label. However, the use of 'pretty names' in Aqualog while learning the system to bridge the gap between user vocabulary and the one used in the ontology is a good attempt to achieve a quality personalisation of the system.

Additional advantage of our system in comparison to Aqualog is that of being capable of analysing queries containing not only instances, relations and concepts, but also values of datatype properties, such as those of type String.

For example, CLOnE QL would give an answer to the question *Which project has start date '11.02.2003.'* having an instance of class Project with defined datatype property of type String with assigned value '11.02.2003.' inside the ontology.

Orakel is a natural language interface to knowledge bases [7] that implements functionality not only for knowledge represented in OWL but also provides means for accessing F-Logic. The key advantage of this system making it different and in a way better than other similar NLIs is support for compositional semantic construction. This means that Orakel is able to handle questions involving quantification, conjunction and negation. However, a mandatory customisation of the system according to the domain-specific knowledge makes it unattractive for end-users. The task of the person in charge of customising the system is to create a domain-specific lexicon mapping subcategorization frames to relations as specified in the domain ontology. Subcategorization frames are essentially linguistic argument structures, e.g. verbs with their arguments, nouns with their arguments, etc.

A very well rated system according to its performance is Librarian [8] - domain specific system for libraries. This system use Wordnet in combination with the dictionary developed w.r.t. the domain specific ontology. The main disadvantage of such a NLI is, similarly to that of Orakel - the dictionary has to be created for each specific domain and most of these customisations of dictionaries have to be performed/supervised by humans.

ONLI (Ontology Natural Language Interaction) [9] is a natural language question answering system used as front-end to the RACER reasoner and RASER's query language, nRQL. ONLI assumes that the user is familiar with the ontology domain but is not required to know how to write queries using the nRQL language. The system will transform the user natural language queries into nRQL query formats. It major difference to other similar systems is that of supporting queries with quantifiers and number restrictions. However, it is not clear from [9] if this system is easily portable and ontology-independent. Moreover, their evaluation shown in [9] doesn't reveal the overall performance of the system, but shows that incorporating quantifiers and number restrictions does not degrade performance of the system.

Querix [10] is another ontology-based question answering system that translates generic natural language queries into SPARQL. In case of ambiguities, Querix relies on clarification dialogues with users. In this process users need to disambiguate the sense from the system-provided suggestions. However, in [10] it is not mentioned what types of questions are handled by the system.

The main advantage of our system in comparison to other similar ones is in its emphasis on robustness, i.e. it gives users the freedom to enter queries of any length and form. It uses heavily reasoning over the ontology, in order to disambiguate and interpret the user queries. It analyse potential relations between concepts and ranks them according to several relevant factors to achieve the most of accuracy. CLOnE QL is also portable, and requires no configuration or customisation when changing the ontology to be used with. It supports com-

positional semantic construction comprising conjunction and disjunction. Last but not least, the query interface is very simple (a Google-like search box), so it can be used without any prior training.

3 Technical Details

CLOnE QL is heavily dependent on the data residing inside the ontology as it preprocesses these to prepare for the process of querying. This preparation is performed during initialisation of the tool, and is aimed to extract domain specific knowledge from the ontology. All relevant terms extracted from the ontology are used to identify key concepts inside the query, while relations between them are retrieved by means of reasoning inside the ontology. Reasoning is also used to solve disambiguity problems to achieve more accurate results. Consequently, the performance of the tool is directly proportional to the quality of the semantic markup in formal descriptions residing inside the ontology: more human understandable descriptions inside the ontology - better result in answering of the queries.

Process of transforming an input query into a formal language query comprises the following steps:

Identifying Key Concepts inside the query. This is performed by producing semantic annotations with respect to a given ontology. We dynamically build a gazetteer from the ontology and use it coupled with a morphological analyser to include all morphological inflections of names and values of properties assigned to ontology resources.

Filtering identified Key Concepts. As sometimes the same part of input text could be annotated with different or similar ontology resources we filter them to avoid those that are of less importance and very often redundant.

Identifying relations between Key Concepts. To achieve most of accuracy of resulting formal queries we retrieve and analyse potential relations (i.e. properties) between identified Key Concepts based on the defined relations in the ontology.

Scoring potential relations. According to the specific factors, using string similarity metrics and a property position in the taxonomy we calculate scores for all potential properties.

Creating SeRQL queries and showing results.

Following sections describe details of each phase.

3.1 Identifying Key Concepts

Semantic annotation is usually the first mandatory step when performing some more important tasks such as semantic indexing, searching, keyword extraction, ontology population and the like. Hence many available tools for producing semantic annotations w.r.t. an ontology such as [11], [12], [13], [14] exist nowadays. However, most of them use static lists for a gazetteer and match only exact text

in documents from that in the list. Our approach differs in that of matching all morphological inflections of the relevant terms by using a morphological analyser in the dynamic construction of the gazetteer lists from the ontologies.

In CLOnE QL we automatically retrieve key concepts from the query by creating ontology-aware annotations over them. These annotations are created based on the assumption that a specific part of a query is referring to a particular resource residing inside the ontology if the lemmas[1] of the two match. A particular ontology resource is identified mostly by its URI, labels, or by a value of some set properties.

Building a Dynamic Gazetteer from the Ontology . To produce ontology-aware annotations i.e. annotations that link to the specific concepts/relations from the ontology, it is essential to pre-process the Ontology Resources (e.g., Classes, Instances, Properties) as well as the set values for the properties and extract their human-understandable lexicalisations. As *rdf:label* property is meant to have a human-understandable value [15], it is a good candidate for the gazetteer. Additionally, labels contain multilingual values, which means that the same tool can be used over the documents written in different languages - as long as that language is supported by the ontology.

However, the part of the Unique Resource Identifier (URI) itself is sometimes very descriptive, making it a good candidate for the gazetteer as well. This part is called *fragment identifier*[2].

As a precondition for extracting human-understandable content from the ontology we created a list of the following:

- names of all ontology resources i.e. fragment identifiers and
- values of all set properties for all ontology resources (e.g., values of labels, values of datatype properties, etc.)

Each item from this list is analysed separately by the Onto Root Application (ORA) on execution (see figure 2). The Onto Root Application is a GATE pipeline of few language processing resources. It first tokenises each linguistic term, then assigns part-of-speech and lemma information to each token.

As a result of that pre-processing, each token in the terms will have additional feature named 'root', which contains the lemma as created by the morphological analyser. It is this lemma or a set of lemmas which are then added to the dynamic

[1] Lemma is the canonical form of a lexeme. Lexeme refers to the set of all the forms that have the same meaning, and lemma refers to the particular form that is chosen by convention to represent the lexeme.The process of determining the lemma for a given word is called lemmatisation.

[2] An ontology resource is usually identified by URI concatenated with a set of characters starting with '#'. This set of characters is called *fragment identifier*. For example, if the URI of a class representing *GATE POS Tagger* is: 'http://gate.ac.uk/ns/gate-ontology#POSTagger', the fragment identifier will be 'POSTagger'.

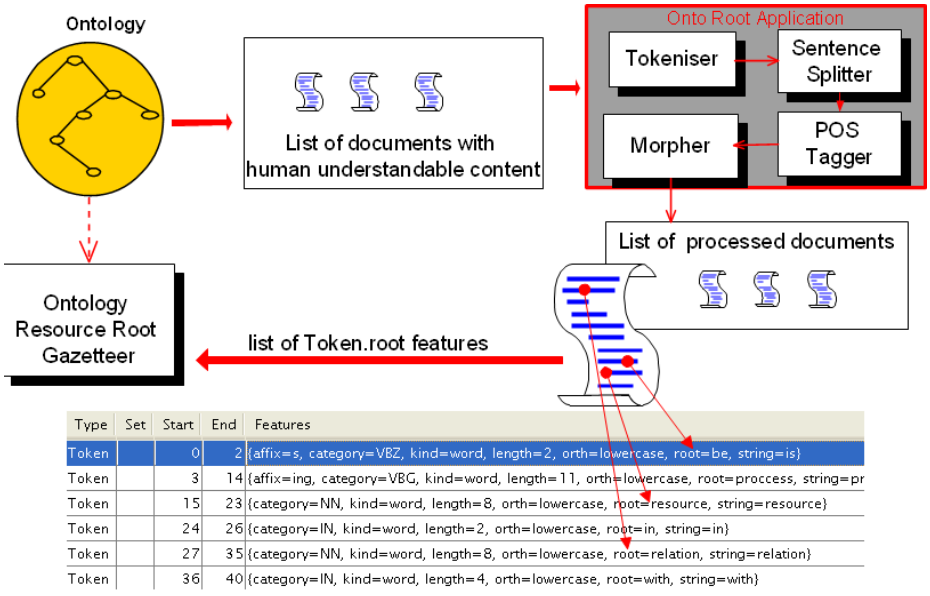

Fig. 2. Building Ontology Resource Root Gazetteer dynamically from the Ontology

gazetteer list (Ontology Resource Root Gazetteer - ORRG), created from the ontology.

For instance, if there is a resource with a short name (i.e., fragment identifier) *ANNIEJapeTransducer*, with assigned property *rdf:label* with values *Jape Transducer* and *ANNIE Jape Transducer*, and with assigned property *rdf:comment* with value *A module for executing Jape grammars*, the created list before executing the OntoRoot gazetteer collection will contain following the strings:

- *ANNIEJapeTransducer*,
- *Jape Transducer*,
- *ANNIE Jape Transducer* and
- *A module for executing Jape grammars*.

Each of the items from the list is then analysed separately and the results would be:

- For *ANNIEJapeTransducer*, *Jape Transducer*, and *ANNIE Jape Transducer* the output will be the same as the input, as the lemmas are the same as the input tokens.
- For *A module for executing Jape grammars* the output will be the set of lemmas from the input resulting in *A module for execute Jape grammar*.

In this way, a dynamic gazetteer list is created directly from the ontology resources and is then used by the subsequent components to annotate mentions

of classes, instances, and properties in the input query. It is essential that the gazetteer list is created on the fly, because it needs to be kept in sync with the ontology, as the latter changes over time.

To enable considering lemmas when annotating query input text against the gazetteer of ontology terms, we use a Flexible Gazetteer. The most important difference between a default Gazetteer and a flexible one is that the latter matches against document annotations, not against the document content itself. In effect, the Flexible Gazetteer performs lookup based on the values of a given feature of an arbitrary annotation type, by using an externally provided gazetteer [16]. As shown on figure 3, the external gazetteer for the Flexible one is set to be ORRG gazetteer, created dynamically.

The output of the morphological analyser in *Onto Root Application* (figure 2) creates features called *root* and adds them to the document tokens (which are annotations of type *Token*). Consequently, we set the *Flexible Gazetteer* in *CLOnE QL Application* (figure 3) to use the values of the *Token.root* features during the annotation process. These features are created when running *Morhper* PR - a morphological analyser over the text-based query.

The output of the *Flexible Gazetteer* is set of annotations of type *Lookup*. To mark these annotations as being related to Ontology Resources we rename them into *OntoRes* using *OntoRes Annotator* - a JAPE[3] based language processing resource. All non-annotated text from the query is processed by *OntoRes Chunk Annotator* and used to create *OntoResChunk* annotations. *KeyWord Annotator* annotates any keywords or keyphrases found inside the query text. Finally, all found annotations are processed by *OntoResAnalyser* processing resource, resulting in the list of SeRQL queries that are than executed and results are being returned to the user. Processing of annotations, filtering key concepts, retrieving and scoring relations between them, creation and execution of the queries are being implemented by *OntoResAnalyser* processing resource. Details follow in next sections.

3.2 Filtering identified Key Concepts

Human language itself is well-known for its ambiguity [17]. It is possible to use the same expression in different context and express the totally different meaning. When identifying Key Concepts, more than one annotation can appear over the same token or a set of tokens, which need to be disambiguated. The most common disambiguation rule is to give priority to the longest matching annotations. We consider an annotation longer than the other one when

- the start offset node is equal or smaller than the start offset node for the other one and
- when the end offset node is greater than or equal to the end offset node for the second annotation.

[3] JAPE is a language for writing regular expressions over annotations, and for using patterns matched in this way as the basis for creating more annotations.

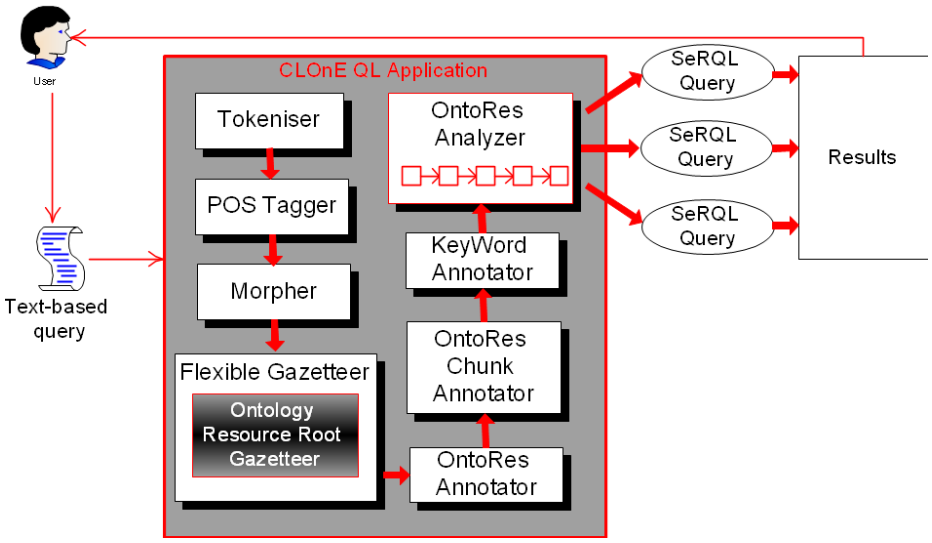

Fig. 3. GATE pipeline of language processing resources in CLOnE QL Application

For example, there is an instance with assigned label with value *ANNIE POS Tagger* inside the GATE domain ontology[4]. This expression comprises the label for the class *POS Tagger* as well, as the class has assigned label *POS Tagger*.

When a document contains the text *ANNIE POS Tagger*, then there will be several annotations indicating that there is more than one resource in the ontology with this name. To illustrate this, we show how this overlapped markup will appear in a GATE graphical viewer (see Figure 4).

Type	Set	Start	End	Features
OntoRes		0	16	{instanceURI=http://gate.ac.uk/ns/gate-ontology#ANNIEANNIEPOSTagger, propertyName=resourceHasName
OntoRes		0	16	{URI=http://gate.ac.uk/ns/gate-ontology#ANNIEANNIEPOSTagger, type=instance}
OntoRes		6	16	{URI=http://gate.ac.uk/ns/gate-ontology#POSTagger, type=class}

Fig. 4. Annotations of type **OntoRes** for input string 'ANNIE POS Tagger'

As the annotation referring to *ANNIE POS Tagger* text inside the document has the start offset smaller than the start offset for the annotation referring to *POS Tagger* text, and the same end offset, we consider it longer and give it a priority. Inside the GATE domain ontology, *ANNIE POS Tagger* is an instance

[4] http://gate.ac.uk/ns/gate-ontology

of the class *POS Tagger* and *POS Tagger* is a class with four instances, one of them being the *ANNIE POS Tagger*. Therefore, in this case, it is possible to disambiguate the mentions to that of the correct instance.

This disambiguation rule is based on the heuristic that longer names usually refer to the more specific concepts whereas shorter ones usually refer to the more generic term.

3.3 Identifying and scoring potential relations

After key concepts are identified we investigate to find any potential relations between them defined inside the ontology. These relations are very important as they add descriptions to the concepts and define their behaviour by adding rules and constraints. The role of relations is of a great importance in the process of Information Extraction (IE) as described in [18]: 'Extraction of relations among entities is a central feature of almost any information extraction task...', where entities can be seen as concepts defined in an ontology.

Information Extraction is a technology based on analysing natural language in order to extract snippets of information. The process takes texts as input and produces fixed format, unambiguous data as output [19]. These data may be further used for different purposes, e.g. in information retrieval applications such as internet search engines. At a higher level of sophistication are systems using semantic annotation in the process of IE. Semantic annotation is about producing metadata and using schema to enable new information access methods so that they enhance existing ones [19]. This information discovered for example in the documents can be connected to formal descriptions, namely, to ontologies.

When retrieving information about arbitrary concepts from ontologies it is crucial to retrieve and understand defined relations i.e. properties for them as it is these relations that affect the behaviour and give limitations to concepts. To retrieve these relations we use reasoning provided by the reasoning component inside CLOnE QL. Retrieved relations are then scored using a combination of three factors. One of them is similarity of the relation's name with the part of the query between identified concepts and is called *similarity score*. The other two relevant factors for scoring the properties are more complex and are based on the property position in the hierarchy of concepts and properties: they are reflected by a *distance score* and a *specificity score*. Next paragraphs provide more information on these.

Similarity score. Similarity of the relation's name with the part of the query (a chunk) between identified concepts is called *similarity score*. The highest score is given to the relation that is the most similar to the chunk. For this comparison we use *Levenshtein distance metrics*. The Levenshtein distance between two strings is the minimum number of operations needed to transform one string into the other, where an operation is an insertion, deletion, or substitution of a single character. Scores varies in range from 0 to 1.

Using string similarity metrics when discovering and ranking property suitability is a straightforward task extensively present in tools for natural language

processing nowadays. To illustrate the way we use Livenshtein distance metrics in CLOnE QL we give the following example. Given a tourism domain ontology and the knowledge base instantiated with common geographical names such as names of cities, countries and continents, and a query 'list cities located in Europe' identified key concepts would be 'cities' and 'Europe', the first referring to a class *City*, and the latter referring to an instance of class *Continent*. The text given in between these concepts (i.e. 'located in') will be further processed and compared with names of all defined properties between identified concepts. As there is the property with name *locatedIn* in this ontology, similarity score between 'located in' and 'locatedIn' will be calculated as 0.8.

Distance score. Distance score reflects the distance of the property from the relevant concepts in the ontology, namely domain and range classes. When designing ontology, concepts are usually presented in the form of hierarchy where the most general ones are at the top, followed by more specific ones at the bottom. For instance, defining that class *Person* has a subclass *Man*, and another one named *Woman*, provides the possibility to define properties with domain or range being class *Man* or *Woman* individually, whereas for relations that could be applied to both concepts the property would be defined on higher level: for *Person* itself. For example, defining property *hasSibling* can result in having *Person* as a range class, whereas for more specific property such as *hasBrother* the range class will be *Man*.

The score that is reflecting the property position according to the position of the relevant domain and range classes in the hierarchy is called *distance score*.

Given property p with $C_{dn}, n = 1, n$ being domain class, and $C_{rm}, m = 1, m$ being range class, the distance score is calculated as a sum of *domain distance score* and *range distance score*:

$D(p) = D_d(p) + D_r(p)$

Domain distance score is the sum of individual distance scores for each domain class of the property:

$D_d(p) = \sum(D(C_{dn}, C_{dnl})), n = 1, n$

$D_d(p) = D(C_{d1}, C_{d1l}) + D(C_{d2}, C_{d2l}) + ... + D(C_{dn}, C_{dnl})$

- where C_{dn} is a domain class of property p,
- C_{dnl} is the lowest class in the hierarchy (i.e. has no subclasses) being a subclass of C_{dn} class, or being C_{dn} class itself in case that C_{dn} has no subclasses, where $n = 1, n$ and
- $D(C_{dn}, C_{dnl})$ is the distance between C_{d1} class and C_{dnl} class

Range distance score is the sum of individual distance scores for each range class of the property:

$D_r(p) = \sum(D(C_{rm}, C_{rml})), m = 1, m$

$D_r(p) = D(C_{r1}, C_{r1l}) + D(C_{r2}, C_{r2l}) + ... + D(C_{rm}, C_{rml})$

- where C_{rn} is a range class of property p,

- C_{rml} is the lowest class in the hierarchy (i.e. has no subclasses) being a subclass of C_{rm} class, or C_{rm} class itself in case that C_{rm} has no subclasses, where $m = 1, m$ and
- $D(C_{rm}, C_{rml})$ is the distance between C_{r1} class and C_{rml} class

Specificity score. Specificity score reflects the position of the property in comparison with other existing properties in the ontology. Similar to the design of concepts inside the ontology, properties that have subproperties usually refer to the generic terms, whereas those that have superproperties refer to more specific ones. For example, while property *hasSibling* is usually defined to have a range of class *Person*, property *hasBrother* could be defined as a subproperty of property *hasSibling* having a range of *Man* only.

Specificity score is determined by the distance of a property from its farthermost subproperty that has no defined subproperties. Its value is normalised by dividing it with the maximum distance calculated for the properties on the ontology level.

Calculating the final score for properties is performed using following equation:

$$Finalscore = similarityScore * 3 + specificityScore * 1 + distanceScore * 1$$

3.4 Creating queries

When all potential relations are scored and ranked, the query using formal language such as SeRQL is dynamically created. Key Concepts usually referring to ontology resources such as classes, instances, properties or values of a properties together with generated scores for each property are used to create relevant query.

Dynamic creation of formal queries makes CLOnE QL flexible and independent, yet easy extendable by any other formal language e.g., SPARQL.

4 Usecases

-supporting open source software teams (TAO)

-supporting access to generic knowledge bases (KIM)

5 Evaluation

1) I think we are actually quite close to getting the thing to answer questions against the KIM KB. 2) I think questionnaire/user satisfaction evaluations are weak (and a major pain to organise). So how about we do some quantitative stuff like for a set of questions:

-measure the amount of interactions with the KIM UI (how many comboboxes one needs to select, how many server-browser round-trips are necessary

for adding all the required restrictions; count the number of words in a NL query that answers the same question.

I have a feeling (though unproven as yet) that our NL interface is more generic than the KIM one, and there may be questions that have answers in the KIm KB but cannot be asked using the UI.

We can also make the point that the NL interface is more natural (exemplifying by showing the actions require for the KIM UI). Though not a measurable statement, any reasonable reviewer would likely agree with us.

which airports does Paris have? http://gate.ac.uk/wiki/Wiki.jsp?page=CloneBacklog

6 Conclusion and Future work

Future work on CLOnE QL includes moving it from being query-based to session-based where we want to include user interaction, similar to that of Aqualog, coupled with mechanism for learning the user's linguistic idiosyncrasies. Current implementation of the system already have an environment to track the process of transforming Natural Language query to SeRQL query. The aim of this tracking is to support user interaction in future. Additionally, this environment would also enable presenting to users how and why system gave the specific result. Users will than have chance to modify the query at some levels of transformations.

Another way of improving the system could be allowing the user to play with the scores or other ways of influencing the decision process. For instance, letting the user reduce the weight on a particular metric when they think it causes some problems in their particular use case.

Acknowledgements. The research for this paper was conducted as part of the European Union Sixth Framework Program projects TAO (FP6-026460).

References

1. Kaufmann, E., Bernstein, A.: How useful are natural language interfaces to the semantic web for casual end-users? In: Proceedings of the Forth European Semantic Web Conference (ESWC 2007), Innsbruck, Austria (June 2007)
2. Tablan, V., Polajnar, T., Cunningham, H., Bontcheva, K.: User-friendly ontology authoring using a controlled language. In: 5th Language Resources and Evaluation Conference (LREC), Genoa, Italy, ELRA (May 2006)
3. Noy, N., Sintek, M., Decker, S., Crubzy, M., Fergerson, R., Musen, M.: Creating Semantic Web Contents with Protégé-2000. IEEE Intelligent Systems **16**(2) (2001) 60–71
4. Popov, B., Kiryakov, A., Ognyanoff, D., Manov, D., Kirilov, A., Goranov, M.: Towards Semantic Web Information Extraction. In: Human Language Technologies Workshop at the 2nd International Semantic Web Conference (ISWC2003), Florida, USA (2003)

5. Lei, Y., Uren, V., Motta, E.: Semsearch: a search engine for the semantic web. In: Managing Knowledge in a World of Networks, Springer Berlin / Heidelberg (2006) 238–245
6. Lopez, V., Motta, E.: Ontology driven question answering in AquaLog. In: NLDB 2004 (9th International Conference on Applications of Natural Language to Information Systems), Manchester, UK (2004)
7. Cimiano, P., Haase, P., Heizmann, J.: Porting natural language interfaces between domains: an experimental user study with the orakel system. In: IUI '07: Proceedings of the 12th international conference on Intelligent user interfaces, New York, NY, USA, ACM (2007) 180–189
8. Serge Linckels, C.M.: Semantic interpretation of natural language user input to improve search in multimedia knowledge base. it - Information Technologies **49**(1) (2007) 40–48
9. Shamima Mithun, Leila Kosseim, V.H.: Resolving quanti?er and number restriction to question owl ontologies. In: Proceedings of The First International Workshop on Question Answering (QA2007), Xian, China (October 2007)
10. 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 (November 2006) 980–981
11. Wartena, C., Brussee, R., Gazendam, L., Huijsen, W.O.: Apolda: A practical tool for semantic annotation. In: In Proceedings of 18th International Workshop on Database and Expert Systems Applications, The Netherlands (September 2007) 288–292
12. Domingue, J., Dzbor, M., Motta, E.: Magpie: Supporting Browsing and Navigation on the Semantic Web. In Nunes, N., Rich, C., eds.: Proceedings ACM Conference on Intelligent User Interfaces (IUI). (2004) 191–197
13. Kiryakov, A., Popov, B., Ognyanoff, D., Manov, D., Kirilov, A., Goranov, M.: Semantic annotation, indexing and retrieval. Journal of Web Semantics, ISWC 2003 Special Issue **1**(2) (2004) 671–680
14. Vehvilinen, A., Hyvnen, E., Alm, O.: A semi-automatic semantic annotation and authoring tool for a library help desk service. In: Proceedings of the first Semantic Authoring and Annotation Workshop. (November 2006)
15. Champin, P.A.: Rdf tutorial. http://www710.univ-lyon1.fr/ champin/rdf-tutorial (April 2001)
16. Cunningham, H., Maynard, D., Bontcheva, K., Tablan, V., Ursu, C., Dimitrov, M., Dowman, M., Aswani, N., Roberts, I.: Developing Language Processing Components with GATE Version 3 (a User Guide). `http://gate.ac.uk/` (2005)
17. Church, K., Patil, R.: Coping with syntactic ambiguity or how to put the block in the box. American Journal of Computational Linguistics **8**(3-4) (1982)
18. Appelt, D.: An Introduction to Information Extraction. Artificial Intelligence Communications **12**(3) (1999) 161–172
19. Bontcheva, K., Cunningham, H., Kiryakov, A., Tablan, V.: Semantic Annotation and Human Language Technology. In Davies, J., Studer, R., Warren, P., eds.: Semantic Web Technology: Trends and Research. John Wiley and Sons (2006)