A Text-based Query Interface to OWL Ontologies

Danica Damljanovic, Valentin Tablan, Kalina Bontcheva

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

Abstract

Accessing structured data in the form of ontologies requires training and learning formal query languages (e.g., SeRQL or SPARQL) which poses significant difficulties for non-expert users. One of the ways to lower the learning overhead and make ontology queries more straightforward is through a Natural Language Interface (NLI). While there are existing NLIs to structured data with reasonable performance, they tend to require expensive customisation to each new domain or ontology. Additionally, they often require specific adherence to a pre-defined syntax which, in turn, means that users still have to undergo training. In this paper we present Question-based Interface to Ontologies (QuestIO) - a tool for querying ontologies using unconstrained language-based queries. QuestIO has a very simple interface, requires no user training and can be easily embedded in any system or used with any ontology or knowledge base without prior customisation.

1. Introduction

Tools for creating, editing and querying ontologies are widely developed to date. However, an initial barrier for using these tools is in the required background knowledge of the field. For querying ontologies, for example, one must be familiar with existing formal query languages such as SPARQL or SeRQL in order to extract useful data. Such languages – while having a strong expressive power – require knowledge of their formal syntax and understanding of ontologies and the way in which they are encoded in languages such as OWL.

To date, many interfaces for querying ontologies are already developed. Some of them are GUI-based and support: a) browsing an ontology, b) constructing a query using predefined templates or c) querying an ontology using formal query languages (e.g., SPARQL). The most popular is Protégé [1] - a platform useful for experts who are familiar with query languages, SPARQL in this case, although they also have to be experienced Protégé users.

KIM - Knowledge Management Platform (Popov et al., 2004) goes one step further in simplifying the semantic search process - it provides an interface for querying knowledge stores by either using predefined query 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 underlying ontology.

According to user preferences the most user-friendly interfaces for accessing data encoded in the form of ontologies is using full-sentence queries (Kaufmann and Bernstein, 2007). However, existing NLI systems tend to be either domain independent (i.e., portable) albeit with lower performance, or more domain-specific (i.e., portable with prior customisation) but with a much better performance. The caveat in the latter case is that customisation tends to be very expensive as it is performed by various experts (e.g., domain experts, language engineers).

The key for bridging the gap between the two extremes would be to carry out customisation automatically. The nature of semantic markup has power to give the sophisticated dimension to NLIs by extracting human-understandable lexicalisations from the ontology automatically. However, for this to happen the quality of semantic information in the formal ontology has to be very high i.e. to contain enough human-understandable labels or descriptions.

In this paper we present QuestIO - a tool for querying a *knowledge store* using natural language. It is domain-independent, easily embeddable and requires no end-user training. As a knowledge store we consider a set of ontologies and the knowledge base containing instances of ontology concepts and relations between them. QuestIO accepts free text query as an input and transforms it into a formal language query (currently SeRQL), see Figure 1. Generated queries are then being executed against the given knowledge store and the result is returned to the user. Ambiguities in the queries are resolved by using reasoning over the ontology, in order to derive all potentially valid interpretations. Finally, there is no customisation necessary for the initialization of QuestIO as all relevant extractions are derived automatically from ontology resources.

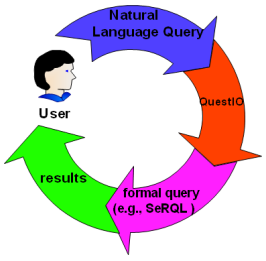

Figure 1: Process of creating formal language queries (e.g., SeRQL) from human language

The paper is structured as follows. In Section 2. we give an overview of existing NLI systems to structured data where we emphasise the main challenges that had to be met dur-

[1] http://protege.stanford.edu/

ing the development of QuestIO. In Section 3. we give details of the QuestIO system, followed by a discussion on QuestIO's language coverage in Section 4. In Section 5. we present the evaluation, and finally we conclude and discuss future directions in Section 6.

2. Related work

Development of NLIs to different kinds of structured data (in particular NLIs to databases e.g. (Hallett, 2006)) has been subject of research for a long time. The main challenges faced during the development of such systems are tightly related to human language itself, which suffers from ambiguity and complexity. A very common way to overcome this problem is to define a *Controlled Language (CL)*. A Controlled Language is a subset of a natural language that includes certain vocabulary and grammar rules that have to be followed. On one hand, a CL provides a simple way to retrieve data without extensive training for the end-user, whilst on the other has less expressiveness than formal languages usually used for accessing structured data.

Although systems that serve as NLIs to relational databases and ontologies have some things in common (e.g., solving the language complexity problem), the main advantage of the latter is the possibility to link the word meanings, inherit the relationships based on the existing structure and deal with ambiguities more efficiently. These advantages, combined with the increasing usage of ontologies, have led to a recent surge in research on NLIs to ontologies. Due to space limitations, here we will discuss only some of the most relevant ones.

Orakel (Cimiano et al., 2007) is a NLI to knowledge bases that besides support for querying knowledge represented in OWL, provides means for accessing F-Logic. The key advantage of this system making it different than other similar NLIs is the 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 might make it unattractive. The task of the person in charge of customising the system is to create a domain-specific lexicon by mapping subcategorisation frames to relations as specified in the domain ontology. Subcategorisation frames are essentially linguistic argument structures, e.g., verbs with their arguments, nouns with their arguments, and the like.

A very well rated system according to its evaluation is Librarian (Serge Linckels, 2007) – a domain specific system for libraries. This system uses Wordnet (Fellbaum, 1998) in combination with a dictionary developed with respect to their domain-specific ontology. This dictionary has to be created for each specific domain and most of the customisation has to be performed or supervised by domain experts.

ONLI (Ontology Natural Language Interaction) (Shamima Mithun, 2007) is a natural language question answering system used as a front-end to the RACER reasoner and RACER's query language, nRQL. ONLI assumes that the user is familiar with the ontology domain but does not need to know how to write queries using the nRQL language. The system will transform the user natural language queries into the nRQL query formats. Its major difference to other similar systems is that of supporting queries with quantifiers and number restrictions. However, from (Shamima Mithun, 2007) it is not clear how much effort is needed for the system customisation for a different domain or query formalism.

Querix (Kaufmann et al., 2006) 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.

In our view, among the many developed NLI systems to ontologies, Aqualog (Lopez et al., 2007) provides the best balance between domain customisation effort and performance. It is also backed by a learning mechanism, so that its performance improves over time, in response to the vocabulary used by the users. However, this system heavily relies on language processing (Lei et al., 2006) and requires syntactically correct sentences.

With a controlled language, such as that used by Aqualog, users can create queries in the form of questions such as: "list hotels in Paris located by the river". Similar or even the same query can be used as input for Web search engines, such as Google. Due to the way search engines work, similar even better results would be given for a much shorter query comprising only the most important concepts, namely: 'hotel Paris river'. This behavior has had a great impact on users, who can be thought of as being 'Googleized', i.e. they are expecting the same simple search-box interface and the same concept-based search behavior from any other language-based search interface. This is why many of the new systems are trying to support concept-based queries, in addition to full syntactically correct ones.

The complementary system to Aqualog is SemSearch (Lei et al., 2006). Whereas Aqualog accepts only full-blown and syntactically correct questions, SemSearch is a concept-based system with a Google-like Query Interface. It requires a list of concepts (e.g., classes, 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 hand, SemSearch does not consider relations between concepts, and neither does it disambiguate in cases when there is more than one relation.

Retrieval of relevant relations is a very important task in our approach, resulting in highly accurate SeRQL queries. Moreover, our approach differs in that it is much more robust with respect to mistakes in the query language, unlike Aqualog. In addition to question-based queries, it also supports concept-based ones such as 'accommodation Rome' which are similar to those supported by SemSearch, only with a simpler and more natural syntactic form.

The main advantage of our system in comparison to the others is in its emphasis on robustness, i.e. it gives users the freedom to enter queries of any length and form. It uses reasoning over the ontology extensively, in order to correctly interpret the user queries, in conjunction with some very

shallow language processing. It analyses potential relations between concept pairs, and ranks them according to several relevant factors to achieve the most accurate interpretation. QuestIO requires no configuration or customisation when changing the ontology to be used with. It supports compositional 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. QuestIO

In our work on QuestIO we aim to address a number of open issues, raised by existing natural language interface systems:

1. Portability without prior customisation.
2. Minimum training for the user.
3. Avoiding use of a controlled language, but allowing users to enter queries of any length and form.
4. Assisting the user in the process of query refinement.
5. Allowing expert users to control the output by providing the mechanism to follow the system's transformations from the input (query) to the output (result), so that they can disagree on system's decisions and refine the query at certain points.

To meet the requirement number 1, we considered automatic domain knowledge extraction from the ontology during tool initialisation (section 3.1.). This extracted knowledge is used at runtime when text-based queries are transformed into SeRQL queries (section 3.2.). While our approach is driven by requirements number 2 and 3, requirements number 4 and 5 are yet to be addressed in our future work, so they will not be discussed further in this paper.

3.1. Extracting domain knowledge automatically

To initialise our system automatically we preprocess the ontology resources (e.g., classes, instances, properties and property values) and extract any human-understandable lexicalisations. To achieve this we first extract a list of the following:

- names of all ontology resources i.e. fragment identifiers [2] and
- assigned property values for all ontology resources (e.g., label and datatype property values)

Each item from the list is further processed so that:

- any name containing dash ("-") or underline ("_") character(s) is processed so that each of these characters is replaced by a blank space. For example, `Project_Name` or `Project-Name` would become a `Project Name`.
- any name that is written in *camelCase* style is actually split into its constituent words, so that `ProjectName` becomes a `Project Name`.

Each item from this list is analysed separately by the Onto Root Application (see figure 2). The Onto Root Application is a pipeline of several shallow language processing modules provided by GATE (Cunningham et al., 2002). It first tokenises each list item, then assigns part-of-speech and lemma (i.e. root) information to each token. It is this lemma or a set of lemmas which are then added to a dynamic gazetteer list (Ontology Resource Root Gazetteer).

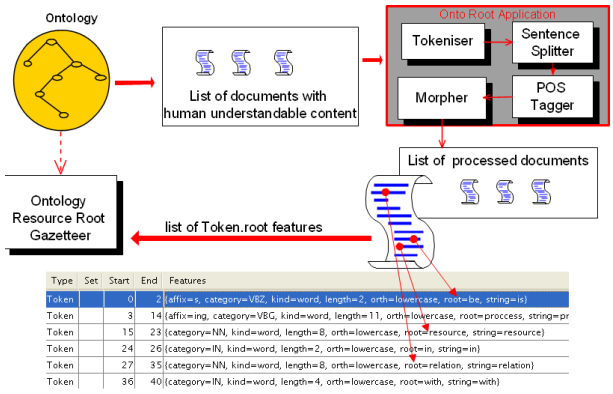

Figure 2: Building a Dynamic Gazetteer from the Ontology

For instance, if there is a resource with a fragment identifier) *ProjectName*, with assigned property *rdfs:label* with value *project names*, the created list before executing the Onto Root application will contain following the strings:

- *ProjectName* as a fragment identifier,
- *Project Name* as split fragment identifier,
- *project names* as the value of *rdfs:label*.

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

- For *ProjectName* and *Project Name* the output will be the same as the input, as the lemmas are the same as the input tokens.
- For *project names* the output will be the set of lemmas from the input, resulting in *project name*.

A dynamic gazetteer list is created directly from the processed ontology resources and is then used by the subsequent components in the process of query interpretation. 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.

The overall performance of QuestIO is directly proportional to the quality of the formal descriptions residing inside the ontology: the more human understandable descriptions are there in the ontology – the better the query interpretation results will be.

[2] An ontology resource is usually identified by an 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'.

3.2. Processing the query

QuestIO is an Information Extraction application, based on the GATE language processing framework (Cunningham et al., 2002). This application accepts a free text query as an input, transforms it to the set of queries expressed in a formal language, e.g. SeRQL, and as an output returns set of results returned after executing these queries against the given knowledge store. Key components of the system are shown in Figure 3.

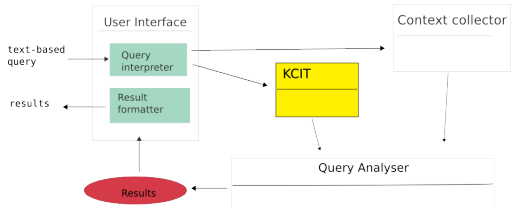

Figure 3: QuestIO diagram

Each user query is interpreted using the *Query Interpreter* in the *User Interface*. It is then analysed by two components, each of which representing a separate GATE pipeline application. Firstly, the Key Concept Identification Tool (KCIT) is identifying key concepts inside the query. Identified key concepts refer to mentions of ontology resources such as instances, classes, properties or property values. To enable considering lemmas when annotating a query against the gazetteer of ontology terms, we process the query with the same language processing resources we used when extracting lemmas in the previous phase (section 3.1.), so that we can then match the extracted lemmas from the ontology resources and the lemmas from the query. In this way, we are matching all existing morphological inflections of the relevant terms.
Secondly, the *Context Collector* collects all words from the query that are not recognised by KCIT, but could be useful in the process of generating the formal query:

- *keywords* such *in, of, from, etc.* – used when analysing the direction of a supposed relation between the two concepts that they connect.
- *keyphrases* usually contain few keywords, or the combination of a keyword and a verb, for example *What are, What is* or *How many*.
- *chunks* – any part of a query that is between two identified key concepts, used later in the relation ranking process.

To give an example, in a query 'What are the countries located in Europe?', KCIT annotates *countries* as a mention of the class `Country`, and *Europe* as an instance of the class `Continent`. *What are* is a keyphrase and *in* is a keyword, both of which will be annotated by the *Context Collector* as they could be used later to help disambiguate the formal queries or to filter results. Additionally, the *Context Collector* would extract the text between all identified key concepts (i.e., chunks), which is in this case only one – *located in*.

Next, the *Query Analyser* uses the identified key concepts from the *KCIT* and all other concepts collected by the *Context Collector* to perform appropriate transformations, formulate SeRQL queries, execute them and send them back to the *User Interface* where the *Result Formatter* renders them in a user-friendly manner.
The Query Analyser, presented in Section 3.2.1. next, combines the key concepts with the other keyphrases, keywords, and chunks, in order to infer any potential relations that are defined between these concepts inside the ontology.

3.2.1. Query Analyser

When all relevant data are collected, the *Query Analyser (QA)* performs the following steps (Figure 4):

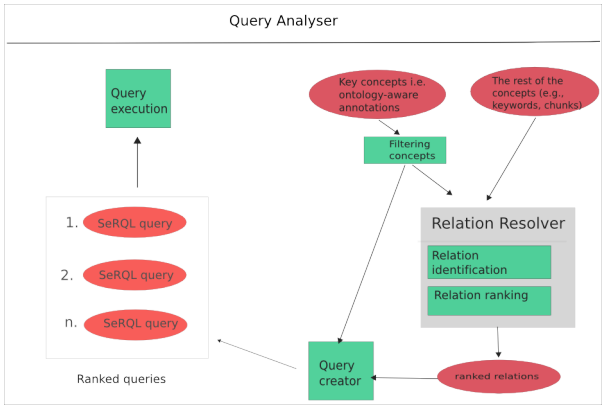

Figure 4: The Query Analyser module

1. *Filtering the identified key concepts.* With human language it is possible to use the same expression in different contexts and express totally different meanings (Church and Patil, 1982). When identifying key concepts, more than one annotation can appear over the same token or a set of tokens, which needs to be disambiguated. The most common disambiguation rule is to give priority to the longest matching annotations. For example, there is an instance with assigned label *ANNIE POS Tagger* inside the GATE knowledge base[3]. The GATE knowledge base contains instances of classes and relations between them based on the GATE domain ontology[4]. *ANNIE POS Tagger* refers to an instance of type *POSTagger*, which has assigned label *POS Tagger*. If a query contains the text *ANNIE POS Tagger* several annotations will be created. One will refer to the class *POS Tagger*, whereas the other one will refer to the instance of that class, namely *ANNIE POS Tagger*.

   As the annotation covering the *ANNIE POS Tagger* string inside the query is longer than the one covering *POS Tagger*, it is given a higher priority. This disambiguation rule is based on the assumption that longer names usually refer to the more specific concepts or instances whereas shorter ones usually refer to more generic terms.

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

2. *Identifying relations between key concepts.* To achieve the best possible query interpretation, we retrieve and analyse potential relations (i.e. ontological properties) between identified key concepts, based on the defined relations in the ontology. These relations are very important as they add descriptions to the concepts and define their behaviour by adding rules and constraints. To retrieve these relations we use ontology-based reasoning provided by the reasoning component inside QuestIO.

3. *Ranking potential relations.* Retrieved relations are then scored using a combination of three factors. One of them is similarity of the relation's name with the chunk and is called a *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*. The next paragraphs provide more information on these.

   The Similarity score reflects the similarity of the relation's name with the part of the query (a chunk) between identified concepts. 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 vary in range from 0 to 1. For instance, if in a query 'list cities located in Europe', identified key concepts would be *cities* and *Europe*, the first referring to the class `City`, and the latter referring to an instance of the class `Continent`, the text given between these concepts (*located in*) will be compared with names of all defined properties between identified concepts. If the property with name *locatedIn* is present in the ontology, the calculated similarity score between 'locatedIn' and 'located In' will be 0.8.

   The Specificity score reflects the position of the property in comparison to other existing properties in the ontology hierarchy. Properties at the top level usually refer to generic terms, whereas those that are closer to the bottom refer to more specific ones. For example, when property *hasSibling* is usually meant to connect two persons, irrespective of their gender, it would have a domain and a range of class *Person*. A property *hasBrother* could be defined as a subproperty of the property *hasSibling* having a range of *Man* only and thus being more specific. In QuestIO we give the higher score to the more specific properties.

   The Distance score reflects the position of the domain and range classes of the property inside the ontology hierarchy. In an ontology, concepts are usually organised in a sub-class hierarchy where the most general ones are at the top, followed by more specific ones lower down. For instance, if unlike in the previous example, the two properties *hasSibling* and *hasBrother* are defined independently, at the same level of the property hierarchy, and the latter has a more specific domain (if we assume that the class `Man` is defined as a sub-class of `Person`), it will be assigned a higher value on the distance score, because in QuestIO properties with more specific domain and ranges are assigned a higher distance score.

4. *Creating SeRQL queries.* When all potential relations are scored and ranked, the formal query in language such as SeRQL is created dynamically. The key concepts referring to ontology resources such as classes, instances, or properties are combined together with the derived properties in order to generate the relevant query. The dynamic creation of formal queries makes QuestIO flexible and independent, yet easy extendable towards any other formal query language e.g., SPARQL.

4. Coverage

Ideally, the range of queries supported by QuestIO is equivalent to or greater than the range of queries supported by formal languages such as SeRQL or SPARQL. In addition, we try not to limit the user by forcing them to construct queries following a pre-defined syntax, as our other goal is to make a tool which will require no training, or alternatively which will be able to interact with the user until they become familiar with the tool and the domain ontology to be queried.

As QuestIO works by recognizing key concepts inside the query before performing other disambiguation and interpretation, and as it creates the formal queries dynamically, the number of concepts in the query is not limited. As long as there are relevant relations between the key concepts in the ontology, the required formal query can be created and the results returned. An example is shown in Figure 5 where in a given query three concepts are identified when ran against the GATE knowledge base: *parameters* - referring to the `ResourceParameter` class, *PR* – referring to the `ProcessingResource` class, and *ANNIE* – referring to the instance of a `GATE Plugin` class. Potential relations are identified between these resources and the appropriate SeRQL queries are constructed.

The other important issue is that QuestIO is not relying exclusively on other words in the query (e.g., keywords), besides the key concepts. As long as it can recognise some key concepts, the remaining parts of the query are used to predict relations and filter the results, but are not required to classify the type of the user query or the type of the formal query that must be generated.

To illustrate this we give an example. If for instance, *Europe* is an instance of the class `Continent` and `Country` is defined class inside an ontology, the queries:*Which countries are located in Europe?* and *countries located in Europe* will in most cases give the same results, regarless of the first query being in the form of a fully-fledged question, and the latter more similar to a concept-based search

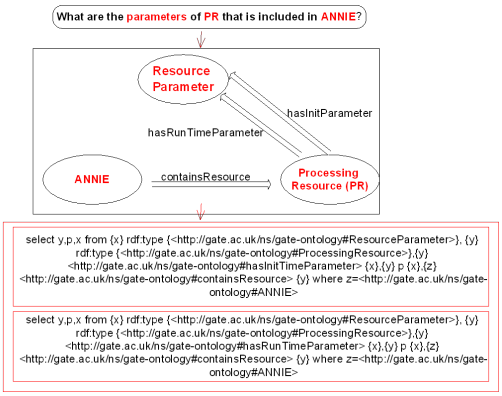

Figure 5: Supporting relative clauses with QuestIO

with important keywords only. Therefore, the same SeRQL query can be generated from a number of different natural language queries, thus providing the user with flexibility.

Furthermore, as long as there are relations between the identified key concepts in the ontology, the appropriate SeRQL query will be formulated, regardless of the number of identified key concepts in the query. For example, in a query *which are the capitals of countries in Southern Europe*, if the key concepts found are: *capitals*, *countries* and *Southern Europe*, the resulting query will include all relations where *capitals* are related to *countries* (e.g., by relation 'locatedIn') and these are in relation with (e.g. by relation 'locatedIn') *Southern Europe*.

Similarly, the order in which key concepts are positioned is not affecting the final result. For example, if a query *List Processing Resources* is run against the GATE knowledge base, all known instances of the class `Processing Resource` will be returned, because *Processing Resources* is identified as a key concept referring to the class `Processing Resource`. *List Processing Resources in ANNIE* would result in listing all processing resources (i.e. instances of class Processing Resource) that are in a relation with an instance *ANNIE*: in the GATE knowledge base, *ANNIE* is an instance of class `GATE Plugin`, and each instance is related to several *Processing Resources* by *containsResource* relation. As QuestIO does not require strict adherence to syntax, the same results would be given for the queries *Processing Resources ANNIE* and *ANNIE Processing Resources*.

Last but not least, our system supports queries including conjunction and disjunction (see Figure 6). These type of queries are processed so that first, those concepts connected with 'and' or 'or' are grouped. Next, relations with other identified concepts are found for each member of the group separately. In this case, SeRQL UNION is used to represent OR, and INTERSECT is used to represent AND.

In this given example, recognised concepts are *parameters* - referring to the class `ResourceParameter`, *ANNIE POS Tagger* - referring to the instance with this label and *Sentence Splitter* - referring to the class with this label. *ANNIE POS Tagger* and *Sentence Splitter* are first grouped. Further on, potential relations between *ResourceParameter* and each member of the previously created group are found,

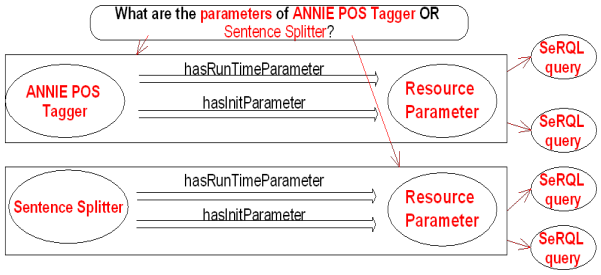

Figure 6: Supporting queries expressing conjunction/disjunction with QuestIO

and SeRQL queries are created accordingly.

5. Evaluation

Here we discuss two kinds of evaluation. The first one is *comparative*, demonstrating the coverage and the advantages/disadvantages of our system compared to Aqualog. The second one is *performance*, where the same queries are executed against two different knowledge bases of different sizes, one being a subset of the other, and demonstrating how the size of the knowledge base affects the query execution time.

5.1. Coverage and correctness

In addition to QuestIO's evaluation on coverage and correctness shown in (Tablan et al., 2008), we extended the same experiment by comparing the overall performance of our system with that of Aqualog (Lopez et al., 2007).

Namely, 36 questions were collected from the GATE user mailing list where users are enquiring about various GATE modules and plugins. These questions were run against the GATE knowledge base, which was created in the TAO[5] project, where we first experimented with learning a domain ontology (Bontcheva and Sabou, 2006) from software artefacts (e.g., GATE source code). This ontology encodes the component model of GATE, the available plug-ins, the types of modules included in each of the plug-ins, the parameters for the different modules, etc. The resulting ontology contains 42 classes, 23 object properties and 594 instances.

Firstly, we filtered the questions so that those enquiring about information that is not in the ontology were excluded (14 out of 36 questions). We then ran the remaining 22 questions against the ontology and categorised results as follows:

- *correctly* answered: the appropriate SeRQL queries were generated.
- *correctly answered after reformulation* of the query: the appropriate SeRQL queries were generated correctly but after the original question was reformulated.
- *partially correct*: the generated queries missed out one of the required constraints, so the answer was less focused. Partially correct answers are scored as 50% correct and 50% wrong.

[5] http://www.tao-project.eu

- *failed* queries: when either no query was generated or the generated query was not correct.

Further, we divided these 22 questions into 2 groups:

1. group: all questions that were malformed or we knew they were not supported by Aqualog, among which there were
   - 1 conjunction query "What are the run parameters of POS Tagger and Sentence splitter?"
   - 1 query with brackets "Does GATE have a coreference resolution component (PR)?"
   - 1 query starting with "How many…"
   - 3 queries not in a form of a full-blown question, for example "I cannot get Wordnet plugin to work".
2. group: full-blown correctly structured questions (16 queries)

Regarding group number 1, out of the 6 questions QuestIO was able to correctly answer 4 queries, 1 question was answered partially, and 1 failed.
Regarding group number 2, we executed all questions using the Aqualog system, and then using QuestIO. The results are shown in Table 1. Reformulating the query in order to be answered with QuestIO did not affect its overall performance, whereas for Aqualog 3 reformulated queries were answered correctly afterwards. For example, *What are the values of the POS Tagger parameters?* was correctly answered by Aqualog when reformulated as *What are the parameters of the POS Tagger?*, whereas both versions of the query were handled correctly by QuestIO.
Among the questions that were answered by QuestIO at least partially correctly, while not being answered by Aqualog, the most common problem was that they were either too long or enquired about information that is partially inside the ontology. For example, the question *is there anybody out there with experiences with MiniPar?* failed to be answered by Aqualog, whereas QuestIO returned details about the MiniPar plugin only.
Several questions of this type were the main reason why our system scored better than Aqualog in overall performance (71.88% for QuestIO and 59.35% for Aqualog). In long and complicated questions, our system was able to recognise at least several concepts and generate SeRQL queries, even though they did not always give the most precise answer.
On the other hand, the Aqualog system has a better interface than QuestIO. For example, in cases when the result is an ontology instance only it would be possible to examine all assigned properties for this instance. In our system, it is only possible to see the name of the instance, and therefore the user has to go to the ontology itself in order to derive more details. Additionally, in the case of disambiguation, Aqualog will prompt the user with a dialogue, whereas our system would automatically derive the result that is ranked best, or in case of several equal scores, it would return all of them without any help from the user.
In future work on QuestIO we will focus on developing a more user-friendly interface supporting user interaction, in order to assist the user in formulating the relevant query.

Table 1: Results of running the same set of queries with QuestIO and Aqualog: *c. correct - conditionally correct (correct after reformulated), p. correct - partially correct*

	QuestIO	*Correct*	**Aqualog**	*Correct*
correct	9 (56.25%)	56.25%	5 (31.25%)	31.25%
c. correct	0	(0%)	3 (18.75%)	18.75%
p. correct	5 (31.25%)	15.63%	3 (18.75%)	9.35%
failed	2 (12.5%)	0%	5 (31.25%)	0%
		71.88%		59.35%

5.2. Portability and scalability

To address the portability aspect, in another experiment we trialed QuestIO with two different knowledge bases of different sizes. One is the Travel Guides Knowledge Base (KB) that contains instances and relations between them from the Travel Guides (TG) Ontology[6]. TG ontology is an extension of the PROTON ontology[7] and contains data about tourism destinations and tourist preferences (Damljanovic and Devedzic, 2008a). In total, this KB has 3194 resources: 318 classes, 86 object properties, 2790 instances.
The core for Travel Guides Knowledge Base contains geographical data such as those about cities, countries and continents (Damljanovic and Devedzic, 2008b). This core was exploited from KIM (Popov et al., 2004) KB that contains general data, specifically about organizations, persons, locations, and has about 40 times more resources than the Travel Guides KB.
Therefore, we prepared a set of queries enquiring about geographical locations and due to the overlap between the two knowledge bases, they give the same results when executed against them. During this experiment, the two knowledge bases have not been changed or customised to work with the QuestIO system. The set of queries chosen were of different level of complexity, where we consider a query *more complex* in comparison to another one if it has more identified key concepts.
As shown in Table 2, the initialization time of QuestIO was around 10 times longer when used with KIM KB, but still within reasonable limits.

Table 2: Initialization time for two knowledge bases of different sizes and execution times for running the same set of queries with QuestIO. Shown times are in seconds.

	TG	**KIM**
Initialization time	22.3	228
Queries	**Execution time**	
countries located in Asia	0.547	5.65
capitals of countries located in Asia	0.203	5.6
capitals of countries in southern Europe	0.109	5.4
which are the political regions in Europe	0.141	11.1
is London capital of any country?	0.625	111
capital country France	0.344	10.6

The execution times were between 10 and 177 times (average: 62) longer when executed with KIM KB. Still, most

[6] http://goodoldai.org.yu/ns/tgproton.owl
[7] http://proton.semanticweb.org

of the queries were executed within few seconds, excluding some exceptional cases. For instance, the query *is London capital of any country?* took 111 seconds as scoring of the properties was not very efficient and several queries were executed returning no results, until it finally found the correct one in the fifth attempt. In future, we will explore other methods to improve the property ranking algorithm, in order to reduce the execution time to a more reasonable level.

6. Conclusion and future work

To summarise, QuestIO transforms queries expressed in natural language to formal ontology languages. This transformation eases the process of querying ontologies for expert users, whereas for non-experts it gives an opportunity to query the knowledge store without first having to learn its particular query language. The main advantages of QuestIO over other similar systems is its support for queries of any length and form and its lack of customisation overhead.

In future work we plan to explore requirements number 4 and 5 from Section 3. Support for user assistance and dialogue-based interaction will help disambiguate queries, which will enable the system to ask users which of the possible interpretations is the one they require. We also plan to extend further the query coverage and to add support for temporal constraints (e.g., give me events in Paris next week) as well as quantitative constraints (e.g., give me hotels in London that cost less than 100 pounds).

Another direction of future work will be extending our knowledge store with annotated documents and implementing an indexing mechanism. This will enable not only answering questions from ontology resources, but also finding the answer inside documents such as Web pages.

7. Acknowledgements

This research was partially supported by the EU Sixth Framework Program project TAO (FP6-026460): www.tao-project.eu.

8. References

Kalina Bontcheva and Marta Sabou. 2006. Learning Ontologies from Software Artifacts: Exploring and Combining Multiple Sources. In *Workshop on Semantic Web Enabled Software Engineering (SWESE)*, Athens, G.A., USA, November.

Kenneth Church and Ramesh Patil. 1982. Coping with syntactic ambiguity or how to put the block in the box. *American Journal of Computational Linguistics*, 8(3-4).

Philipp Cimiano, Peter Haase, and Jörg Heizmann. 2007. 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*, pages 180–189, New York, NY, USA. ACM.

Hamish Cunningham, D. Maynard, K. Bontcheva, and V. Tablan. 2002. GATE: A framework and graphical development environment for robust NLP tools and applications. In *Proceedings of the* 40^{th} *Anniversary Meeting of the Association for Computational Linguistics (ACL'02)*, Philadelphia, USA, Jul. `http://gate.ac.uk/sale/acl02/acl-main.pdf`.

Danica Damljanovic and Vladan Devedzic. 2008a. Applying semantic web to e-tourism. In Zongmin Ma, editor, *The Semantic Web for Knowledge and Data Management: Technologies and Practices*. IGI Global.

Danica Damljanovic and Vladan Devedzic. 2008b. Semantic web and e-tourism. In Mehdi Khosrow-Pour, editor, *Encyclopedia of Information Science and Technology, Second edition*. IGI Global.

Christiane Fellbaum, editor. 1998. *WordNet - An Electronic Lexical Database*. MIT Press.

Catalina Hallett. 2006. Generic querying of relational databases using natural language generation techniques. In *Proceedings of the Fourth International Natural Language Generation Conference*, page 95102. Association for Computational Linguistics, July.

Esther Kaufmann and Abraham Bernstein. 2007. 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.

Esther Kaufmann, Abraham Bernstein, and Renato Zumstein. 2006. Querix: A natural language interface to query ontologies based on clarification dialogs. In *5th International Semantic Web Conference (ISWC 2006)*, pages 980–981. Springer, November.

Yuangui Lei, Victoria Uren, and Enrico Motta. 2006. Semsearch: a search engine for the semantic web. In *Managing Knowledge in a World of Networks*, pages 238–245. Springer Berlin / Heidelberg.

Vanessa Lopez, Victoria Uren, Enrico Motta, and Michele Pasin. 2007. Aqualog: An ontology-driven question answering system for organizational semantic intranets. *Web Semantics: Science, Services and Agents on the World Wide Web*, 5(2):72–105, June.

Borislav Popov, Atanas Kiryakov, Damyan Ognyanoff, Dimitar Manov, and Angel Kirilov. 2004. KIM – A semantic platform for information extraction and retrieval. *Natural Language Engineering*, 10:375–392.

Christoph Meinel Serge Linckels. 2007. Semantic interpretation of natural language user input to improve search in multimedia knowledge base. *it - Information Technologies*, 49(1):40–48.

Volker Haarslev Shamima Mithun, Leila Kosseim. 2007. Resolving quantifier and number restriction to question owl ontologies. In *Proceedings of The First International Workshop on Question Answering (QA2007)*, Xian, China, October.

Valentin Tablan, Danica Damljanovic, and Kalina Bontcheva. 2008. A natural language query interface to structured information. In *Proceedings of the 5h European Semantic Web Conference (ESWC 2008)*, Tenerife, Spain, June.